uc3m | Universidad Carlos III de Madrid

Grado en Ingeniería Telemática
2018-2019

*Trabajo Fin de Grado*

# "Elaboración de data set de ejercicios y materiales de ciberseguridad e interfaz de acceso"


Alberto Casado Medina

Tutor/es

Ana Isabel González-Tablas Ferreres


Leganés, junio 2019

# RESUMEN

Con el presente documento se pretende explicar cómo se ha definido un conjunto de datos y una interfaz de acceso que sirva de punto de partida para todos aquellos profesores y estudiantes que deseen, o bien ampliar sus conocimientos sobre la materia, o bien hacer uso de ellos con fines docentes.

Para ello, ha sido necesario hacer un estudio sobre diversos repositorios existentes, así como investigar aquellas soluciones similares que otras universidades y organismos han llevado a cabo. Además, tras hacer un análisis detallado de todas las tecnologías que pueden llegar a afectar al diseño y desarrollo de la solución se ha decidido lo siguiente.

En un primer lugar, la obtención de los datos se hecho por medio de técnicas de *web scraping*, las cuales los vuelcan a un fichero tipo JSON. Tanto este fichero, como aquel generado al analizar las palabras que contengan los documentos de cada training, es guardado en un sistema de directorios, constituyéndose así el mecanismo de almacenamiento.

En segundo lugar, una vez formado el repositorio, es hora de proporcionar acceso al mismo. Para ello, se ha implementado una interfaz web basada en Django, además de una API REST para desarrolladores. En la web, tras autenticarse, el usuario puede realizar una serie de consultas básicas o avanzadas sobre el contenido del repositorio, de tal forma que el software por detrás lleva a cabo las labores necesarias para encontrar coincidencias entre el texto introducido en la búsqueda y los metadatos y términos almacenados en el sistema de directorios. En la API REST, los desarrolladores, en un primer momento, serán capaces de realizar consultas por cualquier metadato o término, con la condición de incluir únicamente uno de estos parámetros por *request*.

Por último, una vez diseñada e implementada la solución se han llevado a cabo diferentes casos de prueba para detectar posibles errores. Además, se ha estudiado la regulación aplicable y se han dado indicaciones de como se ha gestionado el proyecto, para terminar con la proposición de líneas futuras a implementar.

**Palabras clave:** Web scraper, Django, minado de datos, API REST, learning object repositories.

# AGRADECIMIENTOS

Creo que llevo esperando este momento desde el primer día que comencé el grado en Ingeniería Telemática, allá por 2013. Ha costado, y mucho, pero aquí estoy, redactando las líneas que pondrán punto y final a mi carrera como ingeniero. Estos párrafos que tanto tiempo llevo ansiando escribir, se resbalan por fin entre mis dedos. Nunca había disfrutado tanto haciendo uso de un editor de texto y todo esto es gracias a aquellos que procedo a mencionar.

A mis amigos de la universidad, Pedro, Madru, Joni, Ana, Cristian y Andoni. Por hacer lo difícil, fácil y por todos los buenos momentos que me habéis brindado.

A Chalzs, Mike y Esther, gracias por estar ahí en todo momento, al pie del cañón, parte de este título os pertenece. A mí, me tendréis para siempre.

A mi familia: tíos, tías, primas y abuelos paternos, gracias por interesaros por mis cosas, aunque os hablase en otro idioma. Y, a los que no están, mis abuelos maternos, gracias por escucharme, os debo muchas asignaturas.

A Yvonne, gracias por tu cálido apoyo y tu transmisión de energía. Has estado conmigo desde el principio, me has apoyado en mis momentos más duros y no me has pedido nada a cambio. Este título es tanto tuyo como mío.

A mi hermana, mi guía en la vida, gracias por todos los consejos y ánimos que me has dado. Por las veces que me has liado y por las veces que me liarás. Te quiero más que tú a tu gato.

A mi madre, por su cariño, su preocupación y por haber vivido, si cabe con más intensidad, todos los resultados que he obtenido a lo largo de la carrera. Gracias a ti, podré decir que soy ingeniero.

A mi padre, mi ídolo, solo espero llegar algún día a ser tan grande como tú. Siento todas las decepciones que te haya podido ocasionar, con este título espero resarcirlas. Gracias por tu aliento y tus fuerzas, quiero decirte que me han llegado y que espero que estés orgulloso de mí desde allí donde te encuentres.

Alberto.

# ÍNDICE DE CONTENIDOS

**Índice de ilustraciones** ..... **viii**

**Índice de tablas** ..... **x**

**1. Introducción** ..... **1**

**1.1. Motivación** ..... **1**

**1.2. Objetivos** ..... **2**

**1.3. Organización de la memoria** ..... **3**

**2. Estado del arte** ..... **5**

**2.1. Repositorios** ..... **5**

2.1.1. Merlot ..... 6

2.1.2. Ariadne ..... 6

**2.2. Desarrollos similares de otros centros universitarios** ..... **7**

2.2.1. Linked Open Data University of Münster - LODUM ..... 7

2.2.2. Building Linked Open University Data ..... 8

**2.3. Tipos de ejercicios de ciberseguridad** ..... **8**

2.3.1. Ejercicios tradicionales ..... 9

2.3.2. Ejercicios guiados sobre VM's ..... 9

2.3.3. Capture The Flag (CTF's) ..... 9

2.3.4. Cyberrange y cyberdefense ..... 10

**2.4. Síntesis** ..... **10**

**3. Análisis de la solución.** ..... **11**

**3.1. Descripción general de la solución.** ..... **11**

**3.2. Requisitos del sistema referentes a la interfaz de acceso** ..... **12**

3.2.1. Requisitos funcionales ..... 12

3.2.2. Requisitos no funcionales ..... 13

**3.3. Requisitos del sistema referentes a la elaboración del conjunto de datos** **13**

3.3.1. Requisitos funcionales ..... 13

3.3.2. Requisitos no funcionales ..... 14

**3.4. Estudio de la solución** ..... **14**

3.4.1. Fuentes de origen de los datos ..... 14

3.4.2. Minado de datos ..... 19

3.4.3. Sistemas de almacenamiento ..... 20

3.4.4. Interfaz de acceso ..... 24

**3.5. Síntesis** ..... **27**

**4. Diseño e implementación de la solución** ..... **29**

**4.1. Sistema de ficheros** ..... **29**

4.1.1. Directorios ..... 30

4.1.2. JSON ..... 30

4.1.3. Fichero de términos ..... 32

*4.2. Web scraper* ..... 33
4.2.1. ENISA ..... 33
4.2.2. SEEDLabs ..... 37
**4.3. Análisis de términos** ..... **42**
**4.4. Interfaz de acceso** ..... **43**
4.4.1. Estructura del proyecto Django ..... 43
4.4.2. Gestión de cuentas ..... 44
4.4.3. Interfaz web ..... 46
4.4.4. API REST ..... 64
**5. Despliegue del entorno** ..... **70**
**5.1. Entorno de desarrollo** ..... **70**
5.1.1. Python 3 ..... 70
5.1.2. Django ..... 71
**5.2. Consideraciones para la puesta en producción** ..... **72**
5.2.1. Elección de servidor. ..... 72
5.2.2. Nombre de dominio ..... 73
5.2.3. Certificado SSL ..... 73
5.2.4. Otros aspectos regulatorios. ..... 73
**6. Pruebas** ..... **75**
**6.1. Definición del entorno de plan de pruebas** ..... **75**
**6.2. Web Scrapers** ..... **75**
**6.3. Interfaz web** ..... **77**
**6.4. API REST** ..... **79**
**7. Gestión del proyecto** ..... **80**
**7.1. GitHub como repositorio online** ..... **80**
**7.2. Planificación** ..... **81**
**7.3. Presupuesto estimado** ..... **82**
**8. Marco regulatorio y normativa.** ..... **85**
**8.1. GDPR** ..... **85**
**8.2. Derechos de autor** ..... **86**
**8.3. Sobre el minado de datos - robots.txt** ..... **87**
**9. Conclusiones y líneas futuras.** ..... **88**
**9.1. Consecución de objetivos** ..... **88**
**9.2. Conclusiones personales** ..... **89**
**9.3. Líneas futuras** ..... **90**
**10. Bibliografía** ..... **91**
**Anexo I - Abstract** ..... **95**
**Anexo II – Esquema del sistema de ficheros** ..... **100**
**Anexo III – Requirements.txt** ..... **101**

# ÍNDICE DE ILUSTRACIONES

Ilustración 1 - Traza de ciberataques por segundo y tipo [1] ..... 1
Ilustración 2 - Búsqueda avanzada del repositorio Merlot [9] ..... 6
Ilustración 3 - Búsqueda del repositorio Ariadne [10] ..... 7
Ilustración 4 - Aplicaciones desarrolladas por la universidad de Münster [12] ..... 8
Ilustración 5 - Esquema de la solución ..... 11
Ilustración 6 - Ejemplo de categoría y ejercicio de ENISA [6] ..... 15
Ilustración 7 - Ejemplo de categoría de SEEDLabs [7] ..... 17
Ilustración 8 - Ejemplo de ejercicio de SEEDLabs [7] ..... 18
Ilustración 9 - Arquitecturas de las estructuras JSON [21] ..... 31
Ilustración 10 - Ejemplo del fichero generado por los web scrapers ..... 32
Ilustración 11 - Código de consulta HTTP y conversión de la misma ..... 34
Ilustración 12 - Código para la obtención de los metadatos ..... 35
Ilustración 13 - Código que genera el fichero JSON y el árbol de directorios ..... 36
Ilustración 14 - Código que lee datos, realiza consulta HTTP y la convierte. ..... 39
Ilustración 15 - Código que itera por los distintos enlaces de categorías ..... 39
Ilustración 16 - Código que obtiene parte de los metadatos ..... 40
Ilustración 17 - Código que mina el resto de metadatos ..... 40
Ilustración 18 - Código que inserta los nuevos metadatos y descarga los contenidos .. 41
Ilustración 19 - Árbol de directorios del proyecto Django ..... 44
Ilustración 20 - Código que indica la url por la que acceder al login y su función ..... 47
Ilustración 21 - Variables necesarias para la configuración de social_auth ..... 48
Ilustración 22 - Código que permite la redirección a la autenticación social ..... 48
Ilustración 23 - Código de las vistas de búsqueda básica y de muestra de resultados .. 49
Ilustración 24 - Código correspondiente a la función basic_training_search ..... 50
Ilustración 25 - Código que busca un término dado en los ficheros mainwordfile.txt .. 51
Ilustración 26 - Asignación de las urls de búsqueda simple y mostrar ejercicios ..... 52
Ilustración 27 - Código de la vista de búsqueda avanzada ..... 52
Ilustración 28 - Ejemplo de cómo se visualiza la búsqueda avanzada ..... 53
Ilustración 29 - Código HTML de la vista de la búsqueda avanzada ..... 54
Ilustración 30 - Primer fragmento de código de la función advancedfound ..... 55
Ilustración 31- Segundo fragmento de código de la función advancedfound ..... 56
Ilustración 32 - Código correspondiente a la función get_post_parameters ..... 56
Ilustración 33 - Tercer fragmento de código de la función advancedfound ..... 57
Ilustración 34 - Código correspondiente a la función basic_key_search ..... 58
Ilustración 35 - Ejemplo de búsqueda avanzada con varios criterios ..... 59
Ilustración 36 - Esquema de las lógicas OR y AND ..... 59

Ilustración 37 - Cuarto fragmento del código de la función advancedfound ..... 60
Ilustración 38 - Urls de la búsqueda avanzada y la visualización de los resultados..... 61
Ilustración 39 - Código correspondiente a la función showtraining ..... 62
Ilustración 40 - Visualización del ejercicio en la interfaz web..... 62
Ilustración 41 - Url correspondiente a la visualización de ejercicios ..... 63
Ilustración 42 - Código correspondiente a la función aboutus ..... 63
Ilustración 43 - Url que corresponde a la visualización de la ventana about us..... 63
Ilustración 44 - Código correspondiente a la función usefulinfo ..... 63
Ilustración 45 - Url de la visualización de la ventana de useful information ..... 64
Ilustración 46 - Código correspondiente a la función logout_view. ..... 64
Ilustración 47 - Url de la funcionalidad de cierre de sesión ..... 64
Ilustración 48 - Url para la funcionalidad de la API REST ..... 65
Ilustración 49 - Variables de configuración para la autenticación JWT ..... 66
Ilustración 50 - Código del fichero urls.py del módulo cysectrainingface ..... 67
Ilustración 51 - Código correspondiente a la función traininglist de la API REST ..... 68
Ilustración 52 - Comando curl para solicitar tokens JWT..... 69
Ilustración 53 - Comando curl que solicita una búsqueda ..... 69
Ilustración 54 - Diagrama de gantt del proyecto..... 81
Ilustración 55 - Copyright de los ejercicios de SEEDLabs [7]..... 86
Ilustración 56 - Copyright de los ejercicios de ENISA [6]..... 86
Ilustración 57 - Extensión robots.txt de la web de ENISA [6]..... 87

# ÍNDICE DE TABLAS

Tabla 1 - Posición HTML de los metadatos ENISA ..... 34
Tabla 2 - Posición HTML de los metadatos en la vista de categoría de SEEDLabs ..... 37
Tabla 3 - Posición HTML de los metadatos de la vista de ejercicio de SEEDLabs ..... 38
Tabla 4 - Modelo a cumplimentar para cada caso de prueba ..... 75
Tabla 5 - Caso de prueba 1 ..... 76
Tabla 6 - Caso de prueba 2 ..... 76
Tabla 7 - Caso de prueba 3 ..... 76
Tabla 8 - Caso de prueba 4 ..... 76
Tabla 9 - Caso de prueba 5 ..... 77
Tabla 10 - Caso de prueba 6 ..... 77
Tabla 11 - Caso de prueba 7 ..... 77
Tabla 12 - Caso de prueba 8 ..... 78
Tabla 13 - Caso de prueba 9 ..... 78
Tabla 14 - Caso de prueba 10 ..... 78
Tabla 15 - Caso de prueba 11 ..... 78
Tabla 16 - Caso de prueba 12 ..... 79
Tabla 17 - Caso de prueba 13 ..... 79
Tabla 18 - Caso de prueba 14 ..... 79
Tabla 19 - Presupuesto estimado de los honorarios del docente ..... 83
Tabla 20 - Costes personales del proyecto ..... 83
Tabla 21 - Costes materiales del proyecto ..... 84
Tabla 22 - Coste total del proyecto ..... 84

# 1. INTRODUCCIÓN

Hoy en día existen numerosas plataformas que contienen recursos dedicados a la docencia. Desde los que disponen de *trainings* propios, ideados por miembros de la organización, hasta aquellos que hacen uso de bases de datos comunes para proporcionar al usuario una búsqueda más acorde a sus necesidades. A continuación, se procederá a describir la motivación que ha servido de motor para el desarrollo de este trabajo final de grado. Así como los objetivos del mismo y una breve descripción de lo que se espera reflejar en la memoria.

## 1.1. MOTIVACIÓN

El sector de la ciberseguridad se encuentra en auge actualmente. El número de los ataques cibernéticos aumenta a cada instante, hasta el punto en el que se han ideado numerosas webs, que analizan el tráfico maligno a tiempo real, dejando patente de esta forma el riesgo que suponen.

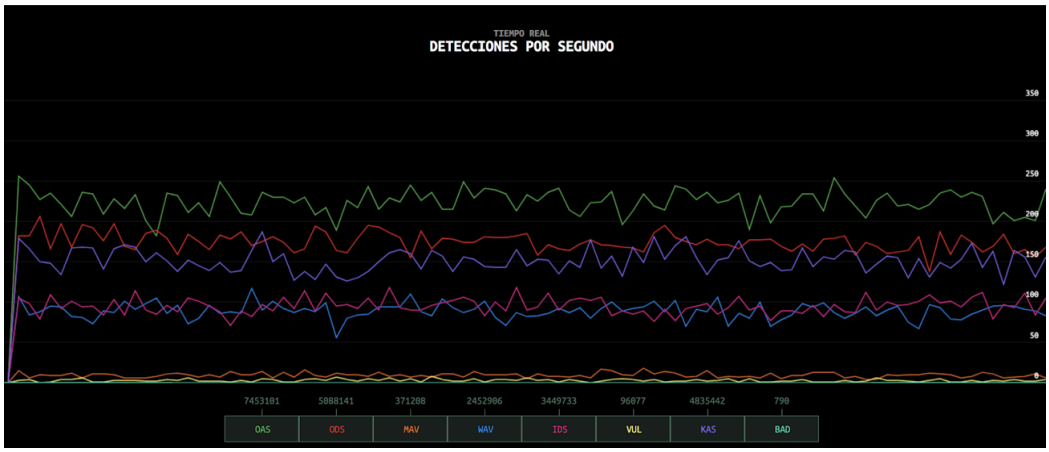



*Ilustración 1 - Traza de ciberataques por segundo y tipo [1]*

La traza que se aprecia en la imagen superior se corresponde a la página web de *Karpesky Labs* [1]. Fue tomada en mayo de 2019, y en cada pico pueden apreciarse las amenazas detectadas por segundo. Es notable la cantidad ingente de ataques y la gran variedad que existe. Si se atiende a su mayor amplitud, es fácil ver que ronda los 1000 ataques por segundo tomando en consideración todas las categorías presentes.

En cuanto a los expertos capaces de hacer frente a los ciberataques, se puede afirmar que no están lo bastante cualificados ni son suficientes. Según un estudio [2] realizado por el Consorcio Internacional de Certificación de Seguridad de Sistemas de Información $(ISC)^{2®}$, en 2022 habrá una brecha de 1,8 millones de empleos sin cubrir en esta área.

Además, es notable el déficit de habilidades presente en los trabajadores del sector, y constituye un grave riesgo tanto para las empresas, los gobiernos y usuarios de la red.

A este suceso, se le añade la proliferación de másteres y cursos de ciberseguridad que están surgiendo como medida paliativa al problema expuesto anteriormente. Sólo en España, se pueden encontrar hasta 60 titulaciones [3]. Además de todas estas, existen actualmente 114 instituciones [4] que imparten formación relacionada con esta materia. Con este hecho, queda reflejado la importancia de la enseñanza en cuestiones relevantes a esta doctrina.

Además de todas estas referencias anteriormente expuestas, se pueden encontrar datos de numerosas certificaciones posibles en este sector. Según la NICCS™, existen 27 principales para desarrollar una carrera en ciberseguridad. [5]

Bajo esta perspectiva, están surgiendo diversos *frameworks* que pretenden catalogar todas aquellas áreas del sector, dando una serie de aptitudes que debe cumplir todo individuo que desee desarrollar su carrera en éste. Las más destacables son los lanzados por NICCS™ e IISP.

En este contexto, nace la motivación de crear un conjunto de datos que simbolice un único, o al menos más general, punto de acceso a todos los recursos disponibles para, de este modo, se facilite la adquisición de conocimientos. Así como la infraestructura que el mismo requiera para su funcionamiento. Con ello, los estudiantes e instructores no tendrán que invertir tiempo en rastrear la web para dar con la formación que buscan, si no que podrán consultar dicho *dataset* y obtener aquellas lecciones que satisfacen sus necesidades.

## 1.2. OBJETIVOS

La finalidad última de este trabajo de fin de grado será elaborar un repositorio de metadatos de ejercicios y materiales de ciberseguridad, así como la configuración de la infraestructura que necesite, siendo escalable a nuevas fuentes de datos. Además, este trabajo tiene como objeto desarrollar una interfaz de acceso a dicho repositorio. Los puntos a tratar para lograrlo, pueden clasificarse en los siguientes:

- Desarrollar de una manera automática un mecanismo que obtenga los metadatos y ficheros de los ejercicios de aprendizaje de las webs proporcionadas, ENISA [6] y SEEDLabs [7]. Estudiar el modelo de almacenamiento más adecuado para dichos datos.

- Analizar el contenido de los ficheros que contienen los materiales con el fin de obtener más información acerca de los mismos.

- Implementar una interfaz gráfica de acceso a los datos previamente obtenidos. Ha de ser segura, robusta y que permita realizar búsquedas básicas y avanzadas. Ha de estar programada de tal forma que sea escalable a largo plazo.

- Implementar un mecanismo que permita a desarrolladores la búsqueda por cualquier metadato. Ha de proveer un mecanismo de autenticación seguro y devolver la consulta en formato JSON.

- La implementación ha de hacerse en todo momento pensando en la futura ampliación que sufrirá el sistema, de manera que evite el hecho de tener que volver a codificar una lógica ya implementada.

- El desarrollo ha de hacerse de manera que el sistema esté protegido mínimamente frente a posibles ataques.

## 1.3. ORGANIZACIÓN DE LA MEMORIA

La estructura de este trabajo se encuentra dividida en diferentes capítulos. A continuación, se procederá a una breve descripción de lo que se espera encontrar en cada uno de ellos.

CAPÍTULO 1: INTRODUCCIÓN

En este apartado se presentarán las motivaciones que han llevado al desarrollo del proyecto. Así como todos aquellos objetivos que pretenden cubrirse con la elaboración del mismo. Por último, se detallará la organización que seguirá la memoria con el fin de facilitar la lectura.

CAPÍTULO 2: ESTADO DEL ARTE

En este capítulo se llevará a cabo un análisis de aquellas soluciones cuyo enfoque sea similar al de este trabajo. Con ello, se reflejará el estado actual en el que se encuentra el panorama y cómo se pretende impactar en él.

CAPÍTULO 3: ANÁLISIS DE LA SOLUCIÓN

En esta sección se describirá punto a punto aquellos aspectos más relevantes a la hora de determinar la solución más apropiada. Se explicará de manera general el

funcionamiento de la misma, así como el sistema más adecuado para llevar a cabo el minado de los datos y su almacenamiento, hasta llegar a las opciones disponibles para proporcionar acceso a estos.

CAPÍTULO 4: DISEÑO E IMPLEMENTACIÓN

En este punto se describirá como se ha decidido llevar a cabo aquellos aspectos mencionados durante el análisis de la solución. Las técnicas necesarias serán explicadas, acompañadas de bastantes documentos gráficos que faciliten el entendimiento del código desarrollado.

CAPÍTULO 5: DESPLIEGUE DEL ENTORNO

En este apartado se detallarán todos aquellos aspectos necesarios para el despliegue del entorno de desarrollo. Se encontrarán unos requerimientos muy precisos que el sistema ha de cumplir para el correcto funcionamiento del software y consideraciones a tener en cuenta para la puesta en producción del sistema.

CAPÍTULO 6: TESTING

En este punto se describirán las pruebas de test que se han realizado a lo largo del desarrollo del programa para comprobar su robustez y su respuesta a fallos.

CAPÍTULO 7: GESTIÓN DEL PROYECTO

En este capítulo se explicarán las herramientas que se han llevado a cabo para gestionar el proyecto y el porqué de su uso. Además, se detallarán aquellos aspectos económicos y de planificación que ha requerido el desarrollo del mismo.

CAPÍTULO 8: MARCO REGULATORIO

En esta sección se detallarán aquellos aspectos referentes a la regulación que puedan afectar al desarrollo del proyecto. Destacando aquellos referentes a la ley de protección de datos y los derechos de autor que le corresponden a los ejercicios.

CAPÍTULO 9: CONCLUSIONES Y LÍNEAS FUTURAS

Por último, se expondrán las conclusiones a las que se ha llegado durante el desarrollo de la solución, junto a líneas futuras que puedan llegar a ser implementadas.

# 2. ESTADO DEL ARTE

El sector de la ciberseguridad se encuentra en auge, tal y como se ha podido leer en la introducción. Existen numerosas instituciones que imparten clases sobre esta temática y expiden certificaciones. Para lo cual, han debido generar previamente materiales en los que apoyar sus explicaciones, que quedan almacenados en las distintas plataformas de cada organización.

La educación es un factor muy relevante en la sociedad, es por eso que no paran de surgir nuevas plataformas de *elearning*, permitiendo a los docentes llegar a un mayor número de público y a los estudiantes, un acceso más sencillo y bajo demanda. Todo esto ha originado que, sin ser el objetivo per sé, se hayan generado multitud de repositorios con ejercicios a los que se puede acceder y clasificar.

Es bajo este contexto, cuando surgen los llamados *learning object repositories*. Consisten en una especie de biblioteca digital en la que los educadores pueden compartir, administrar y reusar recursos educativos. Estos, no solo almacenan el contenido, sino que también los registros de sus metadatos. Existen numerosos estándares respecto a dichos metadatos, sin embargo, o no son lo suficientemente detallados o son demasiado complicados de usar, por lo que su uso no es generalizado. [8] En el ámbito de ciberseguridad se utilizan menos aún pues los materiales en esta disciplina presentan necesidades que no están cubiertas por los actuales estándares (ciberejercicios, CTFs, VMs…). Por estas razones no se estudiarán en este TFG.

Sin embargo, en torno a este concepto, han surgido multitud de organizaciones cuyo fin es reunir el mayor número de materiales de enseñanza para lograr un empleo más eficiente de los mismos mediante su reutilización. A lo largo de esta sección, se comentarán varios repositorios que se han encontrado y son accesibles para todos los usuarios de la web. Así como diversas soluciones que han llevado a cabo distintas universidades de todo el mundo. Para terminar con una explicación de los tipos de ejercicios existentes en cuanto a materia de ciberseguridad.

## 2.1. REPOSITORIOS

Existen numerosas organizaciones que tienen como objetivo constituir una base de datos a la que acudir cuando uno quiera formarse en alguna determinada materia. En la red, pueden encontrarse multitud de repositorios que contienen material docente. En el siguiente apartado se expondrán dos de todos ellos, que disponen de una comunidad que los respalda muy amplia.

### 2.1.1. MERLOT

Comenzó en 1997 como un proyecto del centro de aprendizaje distribuido de la universidad estatal de california (CSU-CDL). Se definen como un sistema que proporciona acceso a materiales de aprendizaje y apoyo en línea, además de herramientas de creación de contenido. Todo ello liderado por una gran comunidad de investigadores, educadores y estudiantes. [9]

En su versión web pueden realizarse búsquedas por hasta 9 categorías diferentes. También permite realizar una búsqueda avanzada con una gran variedad de filtros, tal y como se puede apreciar en la imagen inferior (Ilustración 2). Además, se puede añadir un material o crearlo directamente desde la herramienta que proporciona.

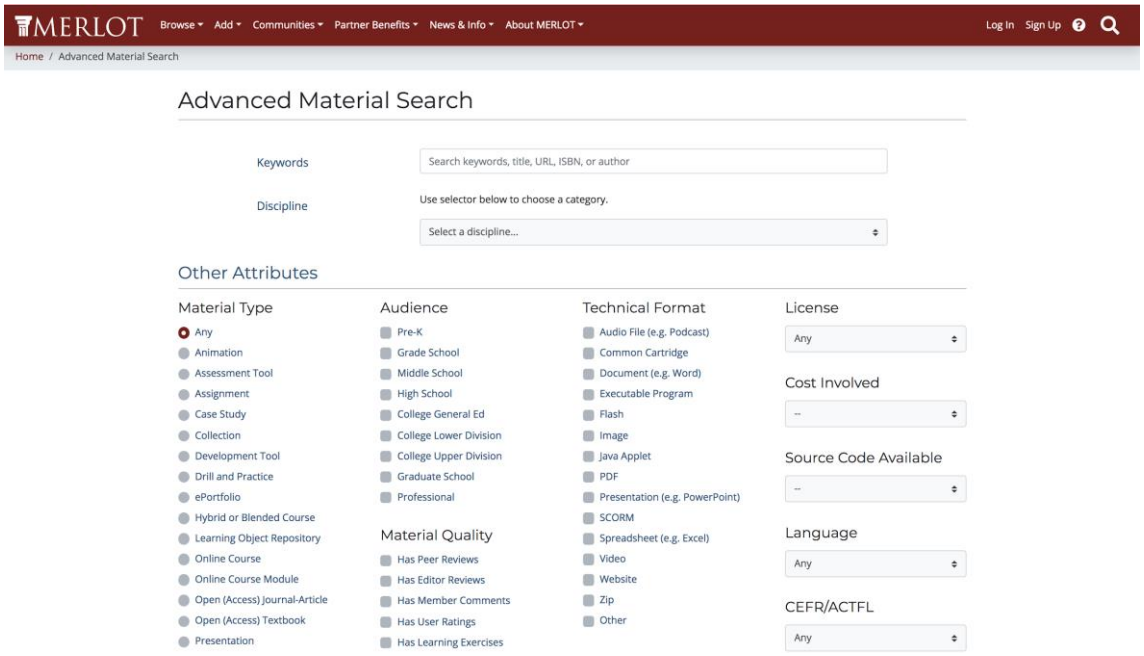



*Ilustración 2 - Búsqueda avanzada del repositorio Merlot [9]*

Su repositorio, está constituido por materiales docentes provenientes tanto de la comunidad, como de sus *partners*, un conjunto de organizaciones que pone a disposición sus recursos educativos.

### 2.1.2. ARIADNE

Se trata de un programa fundado por la comisión europea que busca integrar toda la infraestructura arqueológica presente en Europa para facilitar la labor de investigación a los profesionales, confiando en la comparación, la reutilización e integración de los recursos actuales.

Constituye actualmente un repositorio con unos dos millones de *datasets* recopilados gracias a los integrantes de su consorcio. Entre los que destacan universidades y diversos organismos culturales y gubernamentales.

En cuanto a su portal de búsqueda, presenta una búsqueda muy intuitiva, apoyada en un mapa para indicar la zona geográfica de la cual se quiere obtener datos, o una línea temporal con un gráfico que indica en qué intervalo de tiempo hay más recursos. [10]

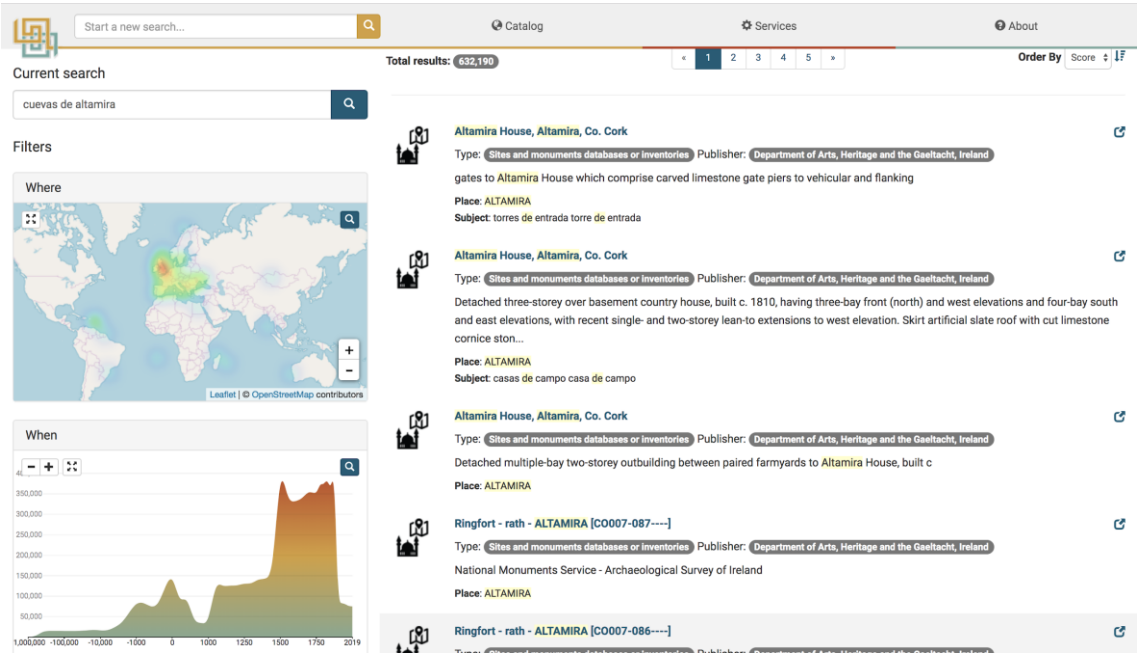



*Ilustración 3 - Búsqueda del repositorio Ariadne [10]*

## 2.2. DESARROLLOS SIMILARES DE OTROS CENTROS UNIVERSITARIOS

Las siguientes soluciones que se presentan han sido llevadas a cabo por universidades mediante *linked data repositories*. Estos consisten en un método de publicación de datos estructurados de tal forma que puedan ser interconectados y más útiles. Hace uso de protocolos de comunicación como HTTP para servir los datos de forma que puedan consultarse de distintas fuentes de datos. [11]

### 2.2.1. LINKED OPEN DATA UNIVERSITY OF MÜNSTER - LODUM

Consiste en un proyecto llevado a cabo por la universidad de Münster mediante el cual se establece una amplia infraestructura para publicar datos de manera enlazada. De esta forma, pretenden otorgar mayor accesibilidad a los datos recogidos y producidos por la universidad que sirvan como fuente de datos para el desarrollo de otras aplicaciones.

En concreto, en esta universidad han empleado ese sistema de datos enlazados para llevar a cabo dos aplicativos. El primero de ellos consiste en un mapa 3D (Ilustración 4 izq.) que representa los edificios de la facultad con una altura proporcional al número de artículos publicados por los investigadores que se encuentran en el mismo. El segundo (Ilustración 4 dcha.) consiste en un sistema que ofrece a los usuarios de la universidad información acerca de los espacios de la misma, así como instrucciones de navegación por los edificios. [12]

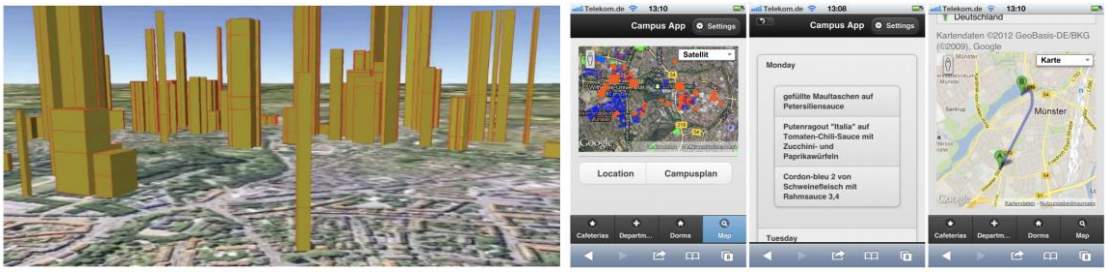



*Ilustración 4 - Aplicaciones desarrolladas por la universidad de Münster [12]*

### 2.2.2. BUILDING LINKED OPEN UNIVERSITY DATA

La universidad de Tsinghua ha llevado a cabo una labor de unificación de diversos *datasets* referentes a las características de los edificios de la universidad, la organización departamental, el personal, fotografías del campus y la programación de cursos.

Con él han desarrollado una aplicación, *CourseFinder*, que proporciona varias formas de buscar información sobre un curso. Dicha búsqueda puede hacerse según varios criterios, permitiendo una fácil navegación entre cursos y conferencias. Además, otorga un acceso sencillo a información de contacto sobre el ponente o profesor.

Otra aplicación que han implementado, basada en el mismo repositorio, ha sido *Campus Assistant*. Es similar a la comentada en el punto anterior de la universidad de Münster, ya que proporciona mapas y navegación por toda la universidad de Tsinghua, haciendo un mayor uso delos *datasets* relacionados con las fotografías de los campus y los edificios de la universidad. [13]

## 2.3. TIPOS DE EJERCICIOS DE CIBERSEGURIDAD

Actualmente existen multitud de variantes de ejercicios de seguridad, desde aquellos basados en contenidos tradicionales, como diapositivas, documentos y prácticas, pasando por los que hacen uso de máquinas virtuales para guiar al estudiante, hasta

llegar a aquellos basados en pruebas, al estilo militar, como captura la bandera. A lo largo del siguiente apartado se explicarán todos los tipos presentes.

### 2.3.1. EJERCICIOS TRADICIONALES

Están basados en materiales clásicos de docencia, en ellos se incluyen documentos con descripciones sobre el ejercicio, videos explicativos, transparencias, prácticas sobre software, etc.

Este tipo de ejercicios constituyen la forma tradicional de enseñanza, pero quedan un poco limitados a la hora de enseñar habilidades prácticas a los interesados en su resolución.

### 2.3.2. EJERCICIOS GUIADOS SOBRE VM'S

Basados en la configuración de una máquina virtual para el desarrollo del mismo. Las máquinas virtuales (VM por sus siglas en inglés) consisten en un software que permite la emulación de un sistema operativo de tal forma que pueden configurarse determinados escenarios que permitan practicar las habilidades explicadas al estudiante.

Es en esta categoría en la que se engloban aquellos ejercicios cuyos metadatos constituirán el repositorio. Concretamente aquellos presentes en las webs de la organización ENISA [6] y los alojados en la del proyecto SEEDLabs [7]. Los primeros, enfocados a perfiles más profesionales, tienen una amplia variedad de máquinas virtuales, cada una configurada de la forma más adecuada para la resolución del ejercicio. Por otro lado, los segundos, orientados sobre todo a estudiantes de grado, hacen uso de una única máquina virtual con el sistema operativo Ubuntu 16.04 para todos los ejercicios.

### 2.3.3. CAPTURE THE FLAG (CTF'S)

Consisten en una serie de tareas que han de llevarse a cabo conforme a su descripción, de tal forma que, a medida que se avanza en su resolución la dificultad va a aumentado. Suelen ser eventos de una duración determinada y una vez han terminado se publican solucionarios con aquellas técnicas empleadas por los participantes. [14]

### 2.3.4. CYBERRANGE Y CYBERDEFENSE

El primero de ellos consiste en un escenario virtual habilitado para entrenamiento de guerras cibernéticas y desarrollo de tecnología de ciberseguridad. Proporciona herramientas que fortalecen la estabilidad, la seguridad y el desempeño de las infraestructuras. [15]

El segundo, consiste de nuevo en otro escenario virtual, pero en lugar de estar orientado a ataques, está orientado a la defensa, enfocándose en prevenir, detectar y responder de manera que no se vea afectada ninguna infraestructura ni información. [16]

## 2.4. SÍNTESIS

Como se ha podido ver, el análisis de las soluciones presentes en el mercado ha arrojado dos cosas en claro. La primera de ellas es la existencia de repositorios que son mantenidos y rellenados por medio de organizaciones a su cargo. La segunda es que, mediante el uso de estructuras de datos enlazados, pueden fusionarse *datasets* que se encuentran distribuidos por la red, en un único repositorio.

Con esto presente, se puede determinar que la solución a la que se va a llegar en este Trabajo de Fin de Grado, va a constituir uno de esos repositorios en los que el contenido del mismo va a ser mantenido y administrado por la organización. Con la descripción de las soluciones basadas en estructuras de datos enlazadas, quería dejarse patente la existencia de este tipo de tecnología, que puede llegar a resultar muy útil cuando el *dataset* está constituido en sí, y no hay que hacerlo desde cero como en este proyecto.

# 3. ANÁLISIS DE LA SOLUCIÓN.

Antes de comenzar con el análisis de la solución es necesario definir aquellos requisitos que impacten en ella. Este ejercicio, permitirá realizar un diseño correcto a tiempo, de tal forma que se eviten imprevistos a la hora de la implementación y se favorezca un desarrollo robusto, escalable y estable.

En el siguiente capítulo se describirá el funcionamiento general del sistema, reuniendo de esta manera, todos aquellos requisitos que compongan el proyecto. Más tarde, se expondrán todas aquellas decisiones que se han tenido que llevar a cabo para lograr el desarrollo de la manera más correcta y eficiente. Para terminar, se sintetizarán las elecciones realizadas con el objetivo de aportar una visión global de cómo se comunicarán todas las tecnologías implicadas.

## 3.1. DESCRIPCIÓN GENERAL DE LA SOLUCIÓN.

El objetivo de este trabajo de fin de grado es realizar una breve recopilación de los datos pertenecientes a todos aquellos materiales de ciberseguridad alojados en las webs de las organizaciones ENISA [6] y SEEDLabs [7], de tal forma que se constituya también una infraestructura escalable a otras fuentes de datos. Con toda esta información se podrá constituir un repositorio sobre el que realizar consultas. El segundo objetivo de este trabajo de fin de grado es desarrollar un acceso para permitir dichas consultas. Éstas podrán ser realizadas de dos formas: mediante API o vía interfaz de usuario.

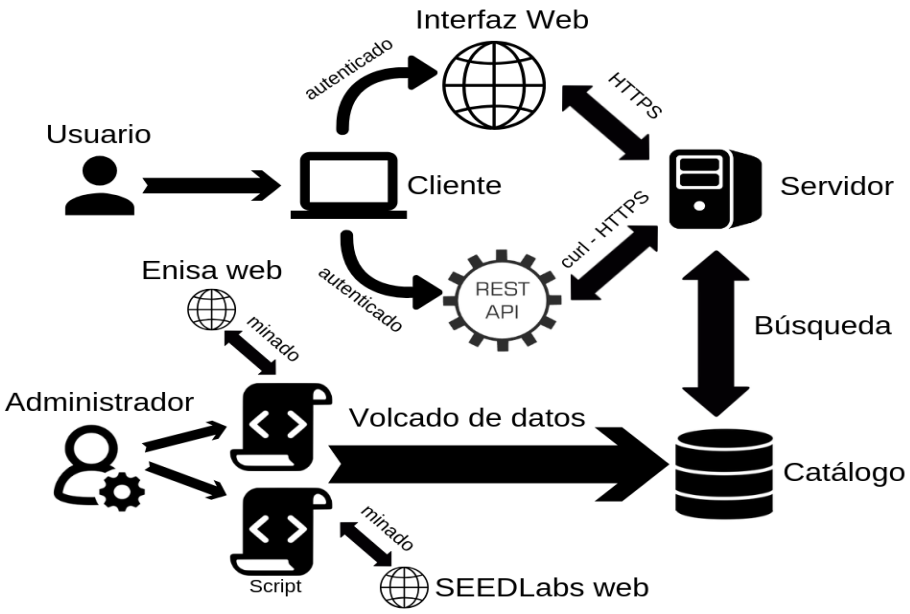



*Ilustración 5 - Esquema de la solución*

En el caso de la interfaz gráfica, una vez el usuario haya proporcionado las credenciales, será redirigido a la vista de búsqueda básica, desde donde podrá llevarla a cabo. La palabra o palabras que introduzca, serán buscadas en todos los metadatos, así como en los términos del repositorio. Aquellos ejercicios que satisfagan la búsqueda serán mostrados, siguiendo la clasificación de por campo o por término. Además de la búsqueda simple, se dará la posibilidad de realizar una búsqueda avanzada, en la que se podrán aplicar operadores lógicos y se podrán seleccionar uno a uno, todos los posibles campos de búsqueda.

En cuanto a la API, su funcionalidad se limitará a una búsqueda simple de ejercicios. La información sobre el cómo utilizarla se encontrará en la pestaña de 'useful information'. Se podrán realizar consultas sobre un único parámetro, el que se desee de los correspondientes metadatos.

## 3.2. REQUISITOS DEL SISTEMA REFERENTES A LA INTERFAZ DE ACCESO

Los requisitos de un sistema son todos aquellos aspectos que ha de cumplir para lograr un objetivo. Éstos pueden ser funcionales o no funcionales. Los primeros se caracterizan por determinar lo que debe hacer el sistema, mientras que los segundos, especifican aquellos comportamientos o características, que han de ser propios del sistema. A continuación, se procederá a la enumeración y explicación de cada uno de los que se han definido.

### 3.2.1. REQUISITOS FUNCIONALES

1. El usuario ha de poder realizar una búsqueda básica. La búsqueda se hará en el repositorio en función de las coincidencias en términos o campos del material.
2. El usuario ha de poder realizar una búsqueda avanzada:
    a. En ella ha de ser capaz de añadir tantos criterios lógicos como desee.
    b. Los operadores lógicos que se proporcionarán serán 'AND' y 'OR' quedando descartados el resto.
    c. Se podrá seleccionar si se desea que el valor introducido esté contenido o sea exacto con respecto al valor que se marca para la búsqueda.
3. El usuario ha de poder visualizar en detalle el ejercicio que desee. Ha de mostrar, si se conoce:
    a. El título.
    b. La descripción.
    c. La fuente.
    d. La duración

    e. Los documentos que lo componen.
4. El usuario que posea cuenta propia del sistema, ha de poder realizar consultas mediante la API, indicando un campo de búsqueda y el valor por el cual desee realizarla.

### 3.2.2. REQUISITOS NO FUNCIONALES

1. El usuario ha de poder autenticarse.
2. El usuario ha de estar autenticado para poder realizar la búsqueda.
3. La búsqueda básica mostrará los resultados en una tabla, con 2 pestañas, la primera para aquellos que coincidan por campo y la segunda por palabra clave.
4. La búsqueda avanzada mostrará en un desplegable todos los campos que contengan los materiales. Salvo si sus valores son comunes a todos los trainings, que los mostrará en *checkboxes*.
5. La búsqueda avanzada ha de disponer de una sección de notas donde se incluirán aspectos que afecten a la misma.
6. La implementación ha de ser escalable. De tal forma que, si nuevas fuentes de datos son añadidas, el código a desarrollar sea mínimo.
7. Es necesario enlazar los documentos propios de los ejercicios a las webs de las que se han extraído, así como el mismo ejercicio a su correspondiente url.
8. La interfaz gráfica ha de permitir el cierre de sesión del usuario.

## 3.3. REQUISITOS DEL SISTEMA REFERENTES A LA ELABORACIÓN DEL CONJUNTO DE DATOS

Como se ha visto en el punto anterior, es necesario definir los requisitos para realizar un buen diseño e implementación. A continuación, se expondrán todos los encontrados para el módulo de la solución referente a la recopilación de datos y su posterior almacenamiento para conformar un repositorio.

### 3.3.1. REQUISITOS FUNCIONALES

1. La información ha de recopilarse de la forma más automática posible.
2. La información obtenida ha de almacenarse de alguna manera robusta.
3. Los documentos PDF's que compongan el ejercicio han de ser analizados en busca de los términos que los conforman.

### 3.3.2. REQUISITOS NO FUNCIONALES

1. La extracción de términos ha de ser escalable a documentos de ejercicios de nuevas fuentes de datos.

## 3.4. ESTUDIO DE LA SOLUCIÓN

En este apartado se detallarán todas las decisiones que se han tomado referentes al desarrollo de la aplicación. La solución consta de varios módulos. Por un lado, es necesario analizar las características de las fuentes de datos de las que obtendremos la información, para más tarde, determinar de qué forma se extraerán los datos que conformarán el repositorio. Después, se estudiará qué sistema de almacenamiento es el más adecuado para lograr el mejor desempeño. Finalmente, La interfaz de acceso ha de ser otro punto a tratar, puesto que simbolizará el contacto directo entre el usuario y el software.

### 3.4.1. FUENTES DE ORIGEN DE LOS DATOS

Para llevar a cabo la solución, lo primero que hay que realizar es un estudio de las webs de las cuales vamos a realizar la extracción de los datos. Para ello, será necesario analizar su estructura y la disposición de sus materiales. Es necesario destacar que, pese a que en un principio se vayan a emplear estos portales, la implementación del repositorio se hará de tal forma que permita ajustarse de la manera más fiable posible a cualquier fuente de datos que se le añada a posteriori y, por tanto, de manera escalable.

#### 3.4.1.1. ENISA

Esta organización [6], cuyas siglas provienen de *European Union Agency for Network and Information Security*, constituye el centro de pericia europeo en cuanto a ciberseguridad. Contribuye en gran medida al desarrollo y promoción de la cultura de la seguridad en las redes y en la información en dicho continente.

Su web dispone de múltiples apartados entre los que se pueden destacar los siguientes:

- TOPICS: recogen una lista con los temas de más actualidad e de interés. Es en esta sección donde se encuentran los materiales de ciberseguridad de los que se van a extraer los datos.
- NEWS: lista todas las últimas noticias referentes a la organización.

- PUBLICATIONS: consiste en un buscador donde se pueden encontrar publicaciones realizadas.
- EVENTS: la agenda de los próximos eventos que organizan.

La url de origen será la siguiente: https://www.enisa.europa.eu/topics/trainings-for-cybersecurity-specialists/online-training-material/. En ella se puede encontrar un listado de trainings de ciberseguridad, clasificados en 4 áreas: *technical*, *operational*, *setting up a CSIRT* y *Legal & Cooperation*.

Si accedemos a cada una de estas categorías, obtendremos un nuevo listado de los ejercicios que la componen y una tabla con la información más relevante de cada uno, tal y como puede apreciarse en la siguiente imagen.

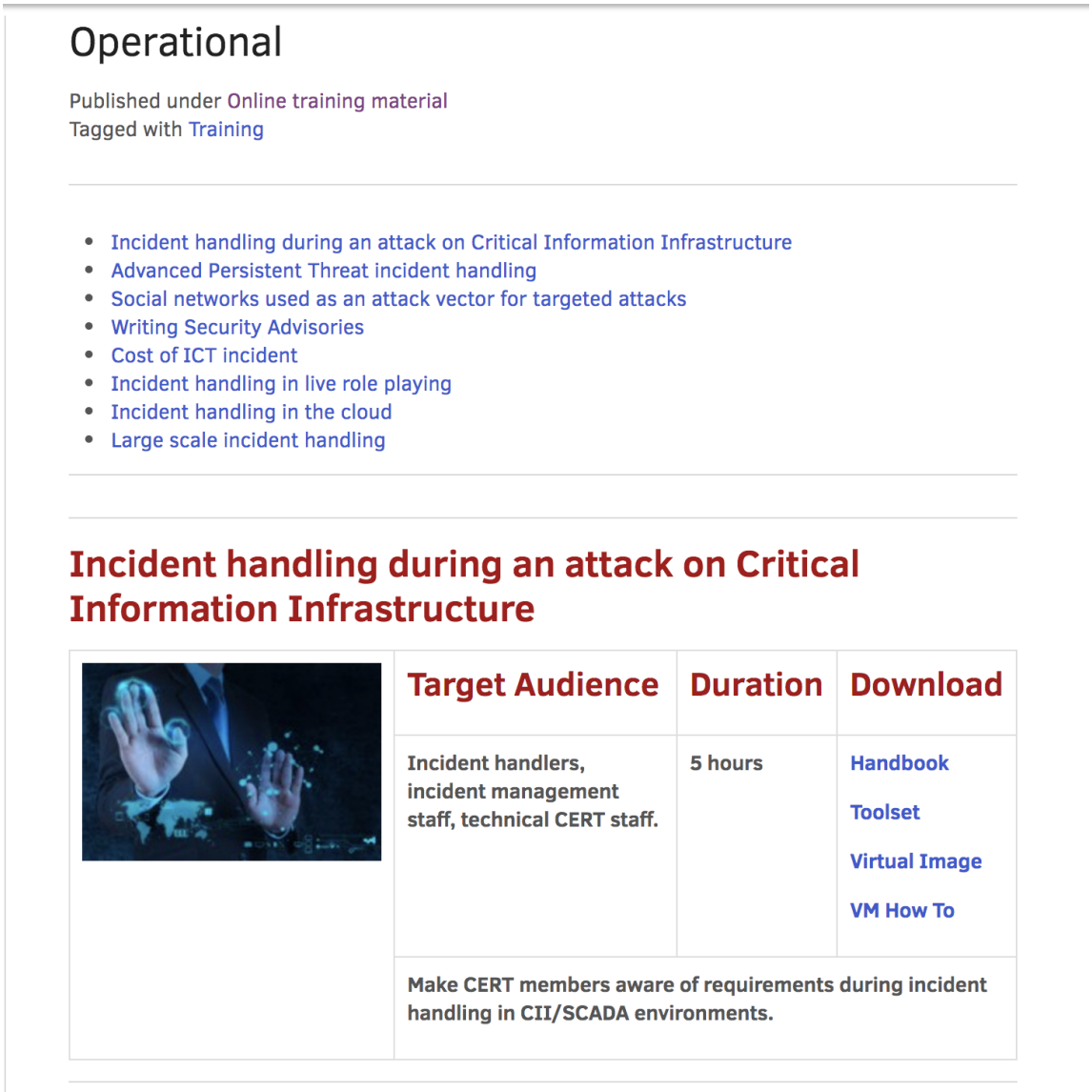



*Ilustración 6 - Ejemplo de categoría y ejercicio de ENISA [6]*

Una vez se ha visto como se disponen los datos, es hora de decidir aquellos que sean de interés. Puede observarse que hay campos de título, público objetivo, duración y descripción que han de ser obligatorios capturarlos por lo que suponen en sí mismos. Sin embargo, la imagen que los acompaña no aporta valor al análisis del material por lo que se ha decidido no incluirla en el *dataset*. Por último, en cuanto a los archivos descargables, se almacenará tanto su *url* como su nombre, lo que permitirá realizar búsquedas por dichos valores.

#### 3.4.1.2. SEEDLABS

Se trata de un proyecto llevado a cabo por la Universidad de Siracusa, en Nueva York. Está liderado por el profesor Wenliang Du con colaboración de sus alumnos. Reúne una serie de laboratorios en Ubuntu que se focalizan en diversas áreas de la ciberseguridad. [7]

La web en la que están alojados los ejercicios es bastante sencilla, consta de las siguientes partes:

- HOME: se puede encontrar información relevante al proyecto.
- SEED LABS: es en esta pestaña donde se encuentran los ejercicios de los que vamos a extraer la información.
- LAB SETUP: se explica la configuración que ha de seguirse para el desarrollo de los ejercicios.
- DOCUMENTATIONS: se recogen todos aquellos documentos, manuales y publicaciones que pueden ser de interés.
- WORKSHOPS: aloja información acerca de los siguientes workshops que realizarán.
- ABOUT: muestran la información relevante al proyecto y a las personas que colaboran en él.
- NEWS: lista todas aquellas noticias de interés sobre el proyecto.

La url de origen será la siguiente: http://www.cis.syr.edu/~wedu/seed/Labs_16.04/ . En ella se podrá encontrar un listado de categorías en las que se han dividido los trainings: *Software Security Labs, Network Security Labs, Web Security Labs, System Security Labs, Criptography Labs y Mobile Security Labs*. Dentro de cada una de estas categorías, se podrá encontrar una lista de los trainings que la componen, tal y como se puede apreciar en la siguiente imagen.

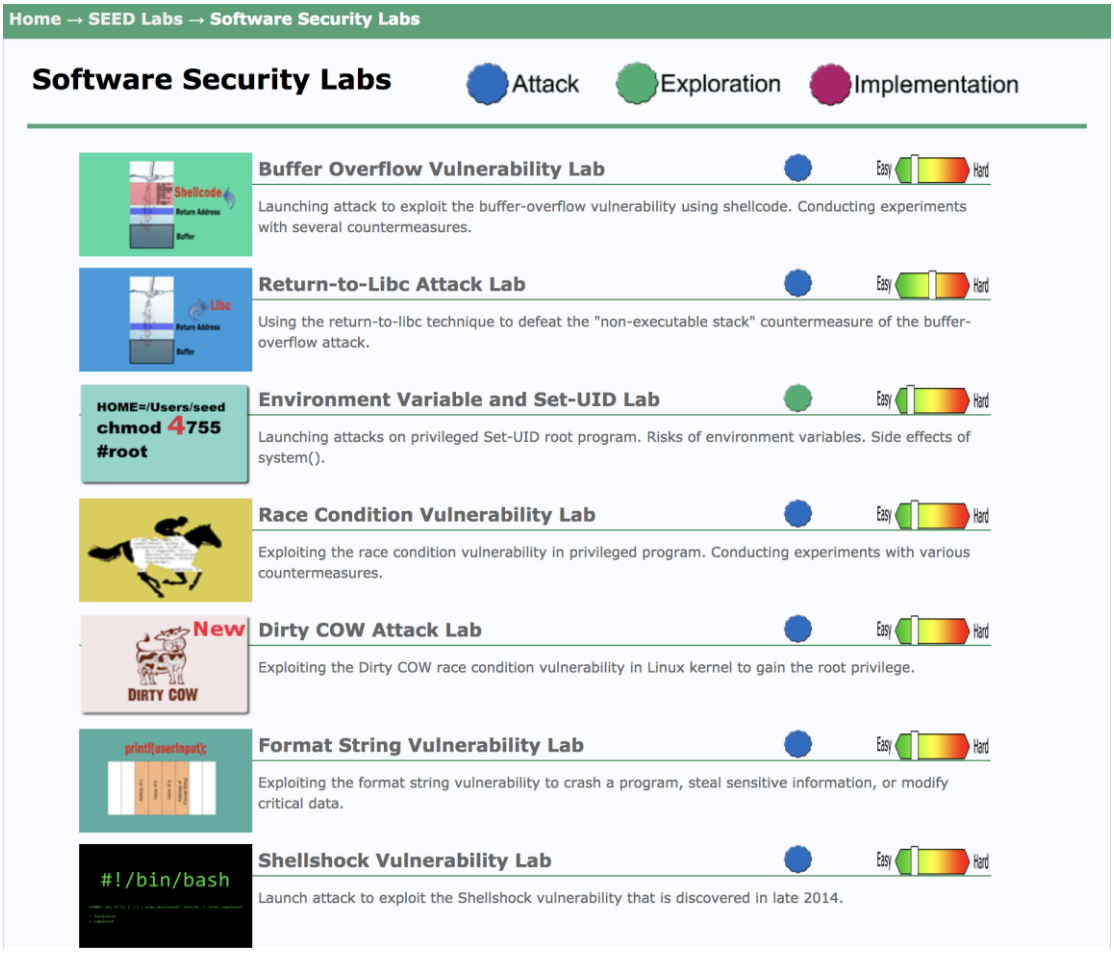



*Ilustración 7 - Ejemplo de categoría de SEEDLabs [7]*

Los campos a relevantes de cada ejercicio en esta vista pueden ser el título y la breve descripción a simple vista. Sin embargo, si nos fijamos en los iconos situados a la derecha del título, hacen referencia al tipo de ejercicio y a su dificultad, por lo que será interesante tratar de obtener esos datos de alguna manera también.  Por último, la imagen de cada training no aporta nada sustancial a los datos por lo que se ha decido no tomarla en consideración.

Una vez analizada la vista en la que se muestran el conjunto de materiales de cada categoría, es hora de analizar la vista de cada uno en detalle. En la captura inferior (Ilustración 8) se podrá observar la visualización de la web.


*Ilustración 8 - Ejemplo de ejercicio de SEEDLabs [7]*

Tal y como se puede comprobar, posee numerosos campos que son atractivos a la hora de lograr almacenarlos. Por un lado, se puede apreciar como la descripción se ha ampliado, por lo que se tendrá que añadir a la recogida en la vista anterior. Además, en la sección de "*Lab Tasks*", se puede ver como existe un hipervínculo a la descripción del material, por lo que se tratará de almacenar ambos valores, el nombre del hipervínculo y el link en sí. Esto mismo sucede con los vídeos. En cuanto al tiempo estimado, al tratarse precisamente de una estimación bastará con recoger aquel que sea mayor, es decir, el presente en el segundo punto de la lista. Los ficheros necesarios serán añadidos a la lista de los archivos y *urls* empleada anteriormente en la sección de "Lab Tasks" puesto que afectan de igual manera al entendimiento del ejercicio. Por último, las lecturas sugeridas, no serán tomadas puesto que no representan gran información y podrían dispersar las búsquedas en el repositorio.

### 3.4.2. MINADO DE DATOS

En este subapartado se detallará el estudio que se ha realizado sobre los mecanismos para la obtención de datos. Es necesario recalcar que el repositorio estará formado por 2 tipos de información:

- Metadatos: constituidos por todo aquel dato relevante que se proporcione en la web acerca de cada material de ciberseguridad.
- Términos: estos datos surgirán al llevar a cabo un análisis de las palabras presentes en los documentos descriptivos del ejercicio en cuestión.

Una vez se han determinado las dos clases de información que se van a tratar, es necesario ver de qué forma se va a proceder a su obtención.

#### 3.4.2.1. METADATOS

La obtención de los metadatos se realizará mediante un minado de estos por medio de técnicas de *web scraping*. Las cuales consisten en extraer información de sitios web mediante la simulación del comportamiento humano en la *World Wide Web*. [17] Para ello, es necesario analizar el código HTML que conforma la página y programar el software de forma que obtenga los datos de interés.

La ventaja que presenta esta técnica es que una vez invertido el esfuerzo de codificar la lógica, la obtención de los datos es automática y, en caso de que en el futuro aumente el contenido de la página (manteniendo el formato), no habrá que dedicar más tiempo en la obtención de los mismos, simplemente bastará con ejecutar de nuevo el software. Es importante recalcar aquí, que la ejecución periódica del mismo no ha de realizarse en intervalos de tiempo muy cortos. Esto es debido a que muchos servidores emplean mecanismos para detectar tráfico recurrente o sospechoso, y pueden llegar a bloquear la IP desde la que se realiza la consulta, lo cual implicaría un gran inconveniente a la hora de actualizar los nuevos datos.

Sin embargo, el hecho de automatizar un proceso que se basa en una determinada estructura, puede no llegar a ser muy robusto. En el caso de que nuevos campos surjan o cambie la estructura de la web, se vería afectado el funcionamiento, siendo necesario una modificación. Por lo que requiere de una cierta supervisión para determinar si se están obteniendo los resultados deseados.

Existen numerosos programas y herramientas que posibilitan está acción. Desde aquellos gestionados por empresas, que proporcionan una interfaz gráfica para facilitar su uso, hasta módulos y librerías de programación, destinadas a que cada desarrollador implemente el minado de la forma que más le convenga.

En este trabajo, se ha optado por omitir los *softwares* de terceros y proceder a una implementación propia, mediante el uso de módulos de Python 3, concretamente "*wget"* y "*Beautiful Soap"*. Para ello ha sido necesario un análisis de las webs objetivo que se detallará en el apartado de diseño.

#### 3.4.2.2. TÉRMINOS

Las palabras de los ejercicios han de extraerse con el fin de lograr un repositorio más completo. Analizando las palabras presentes en los documentos de cada material, se podrá hacer una categorización mejor del mismo y, por tanto, lograr una mayor precisión en las búsquedas que se realicen.

La extracción de palabras se acometerá mediante el uso de librerías de Python 3. Entre las que destacan "*PyPDF2*", la cual permite la lectura de un fichero PDF, "*textract"*, que extrae el texto de la lectura previamente hecha y "*nlkt*" que contiene un amplio diccionario de palabras frecuentes que no aportan valor al análisis.

### 3.4.3. SISTEMAS DE ALMACENAMIENTO

A continuación, se procederá con el análisis que determine el sistema de almacenamiento más apropiado para almacenar estos dos tipos de información. Al final de cada uno de los subapartados de los que consta este punto, podrá encontrarse una breve explicación de las causas por las cuales se ha decidido elegirlo o descartarlo.

Hoy en día, las alternativas para almacenar datos son ingentes. Desde bases de datos no relacionales en las que prima la escalabilidad y el manejo de gran cantidad de información, pasando por bases de datos relacionales, cuya máxima es cumplir el principio ACID, llegando hasta los sistemas de ficheros, donde la información es organizada en árboles de directorios. La finalidad de esta sección es debatir entre todas estas opciones y concluir cuál es la más adecuada para el desarrollo de nuestra aplicación.

Como ya se ha definido antes, la aplicación a la que se refiere este trabajo necesitará almacenar una serie de campos referentes a características que poseen los materiales

de las páginas webs a las que se conectará, junto a los términos que los conforman. Este almacenamiento a su vez, tiene que ser flexible, debido a que cabe la posibilidad de que en el futuro se añadan nuevas fuentes de datos o se modifiquen las ya presentes.

Cuando se piensa en el almacenamiento de información, instintivamente se viene a la cabeza el emplear una base de datos relacional, puesto que es lo más común. Sin embargo, no siempre es la mejor de las opciones. En este caso, se va a demostrar por qué es más recomendable emplear un sistema de almacenamiento basado en ficheros, frente a los modelos de bases de datos relacionales y no relacionales. Para ello, se enumerarán las características principales junto a las ventajas y desventajas que se les atribuyen a cada uno de los sistemas.

#### 3.4.3.1. BASES DE DATOS RELACIONALES

Consisten en un conjunto de tablas relacionadas mediante claves, cuyos atributos están dispuestos en columnas. Las filas componen tuplas y representan valores característicos del objeto que se desea representar. Se fija de esta forma un modelo relación-entidad.

Las ventajas que presentan las bases de datos relacionales son muy diversas. Las más destacables son:

- Redundancia de los datos: emplean mecanismos que modifican todos los archivos cuya información sea idéntica.
- Compacidad: no existe duplicación de ficheros.
- Rapidez: frente a usuarios gracias a su organización por campos, registros y tablas accesibles mediante autorización.
- Concurrencia sin fallos: permiten el acceso compartido de usuarios sin comprometer la consistencia de los datos.

Su eficacia viene determinada en bastante medida por la memoria y espacio en disco que posean. Sin embargo, pese a todas estas aptitudes, son dueñas también de un amplio rango de desventajas:

- Sistemas complejos: que requieren memoria y disco suficientes.
- Lentitud: frente a determinadas aplicaciones.
- Requieren mantenimiento.

Estos factores hacen que se necesiten de una infraestructura dedicada para lograr un correcto funcionamiento, además de personal cualificado con conocimientos suficientes

como para gestionar todas las incidencias que puedan surgir. La arquitectura de la aplicación, no permitirá a los usuarios hacer peticiones de escritura, por lo que esa potente funcionalidad propia de las bases de datos, quedaría en desuso. Además, el hecho de almacenar los términos, implicaría llevar a cabo consultas con expresiones regulares, algo para lo que está más preparado otro tipo de sistemas. Si a estos hechos se le añaden, los esfuerzos que supondría el modificar las tablas para añadir la información de nuevos campos, se puede concluir que no es la tecnología más apropiada para este proyecto.

#### 3.4.3.2. BASES DE DATOS NO RELACIONALES

Este modelo es el más moderno. Se puede decir que surgió de la necesidad escalar las funcionalidades de las clásicas bases de datos relacionales dado el aumento exponencial de datos que debían almacenarse. Cambian el esquema de entidad-relación propio de la tecnología expuesta en el punto anterior por mapeos de columnas, clave-valor o grafos (donde la información y sus relaciones se representan como aristas del mismo, de tal forma que se pueden recorrer haciendo uso de la teoría de grafos) [18]

Las ventajas [18] que presenta este tipo de tecnología son las siguientes:

- Necesita pocos recursos: no requieren apenas computación, a diferencia que los explicados anteriormente.
- Escalabilidad horizontal: se puede mejorar su eficiencia fácilmente añadiendo más nodos e indicando cuáles están disponibles.
- Manejo de gran cantidad de datos: al emplear una estructura distribuida.
- Sin cuellos de botella.

Aunque son las más actuales, no están exentas de desventajas, entre las que podemos encontrar:

- No atomicidad: es sustituida por consistencia eventual, lo que garantiza que, si no se realizan nuevas actualizaciones de un dato, finalmente todos los accesos a ese dato devolverán su último valor. Por lo que la consistencia lograda puede no ser del 100%.
- No están estandarizadas: Se pueden encontrar diversos tipos de bases NoSQL.
- Sistema operativo: Están optimizadas para su uso en Linux.
- Compatibilidad con SQL: al haber varios tipos, hace que no todas sean compatibles con dicho lenguaje de consulta.

Si se atiende a las características de este tipo de almacenamiento, podría elegirse sin dudas como la solución más válida. Sin embargo, hay que resaltar, que el acometido de este proyecto en principio, no tiene como objetivo manejar una ingente cantidad de datos. Además, presenta el mismo déficit que en el caso anterior, el usuario no realizará peticiones de escritura, perdiendo de esta manera la funcionalidad principal de una base de datos. Es por esto que, se ha decidido descartar los sistemas NoSQL (también conocidos así) puesto que no se aprovecharía todo el rendimiento que son capaces de dar.

Una vez estudiadas las ventajas y desventajas de este tipo de sistema de almacenamiento, se procederá a hacer lo propio con el último tipo de almacenamiento: el sistema de ficheros.

#### 3.4.3.3. SISTEMA DE FICHEROS

El último de los modelos a estudiar consiste en la creación de un sistema de almacenamiento mediante ficheros. En estos, la información se encuentra indexada en directorios y archivos.

En cuanto a las ventajas que poseen los sistemas de ficheros se pueden destacar varios puntos:

- Escasa complejidad: que requieren para acceder a su contenido.
- Organización e individualización: son fácilmente organizarles según la aplicación para la que se diseñen. En este tipo de almacenamiento, la información se estructura en carpetas y subcarpetas.
- Control de accesos: sólo será accesible por aquellos usuarios que dispongan de una copia en su sistema o si se les proporciona acceso a la misma mediante red.
- Redundancia: únicamente modifican el fichero que haya sido editado y guardado.
- Espacio de almacenamiento: no requieren demasiado espacio de información, más allá del empleado para el almacenamiento de los ficheros.

Es fácil ver como todas estas propiedades se ajustan a la perfección a las características de la aplicación. Sin embargo, también poseen una serie de desventajas que no se deben omitir:

- Tiempo de búsqueda: varía en función del tamaño, por lo que no es recomendable tener sistemas de ficheros muy extensos.

- Edición y envío de información: presenta más dificultades y pueden traer consigo una desorganización.

Los inconvenientes de este sistema de almacenamiento son mínimos, si se atiende a la funcionalidad que ha de tener el software a desarrollar en un principio. Por un lado, nos encontramos que el tiempo de búsqueda varía en función del tamaño, aspecto que llegará a repercutir en un futuro a medida que se aumente las fuentes de origen. Sin embargo, si llegase a tal punto, podría migrarse el sistema de ficheros a una base de datos no relacional, pues en ese punto sí sería más eficiente emplearla. Por otro lado, el referente a la edición no es relevante puesto que el usuario no llegará a editar ningún fichero.

En cuanto a las ventajas, se pueden dilucidar suficientes como para decantarse por esta tecnología como sistema de almacenamiento. Pasando por encima de la escasa complejidad que requiere, algo que siempre es de valorar, permite una buena organización e individualización, aspecto que será muy relevante a la hora del diseño de la aplicación, tal y como se podrá leer en el punto de diseño. Además, por otro lado, permite tener la información sin necesidad de un flujo de peticiones adicional por la red, reduciendo de esta forma la latencia del servidor en el que se aloje. El hecho de que no necesiten de almacenamiento adicional, simplemente aquel destinado para la contención de los ficheros, hace que sea un punto fuerte ya que otorgará menos restricciones a la hora de querer instalarlo en un host con peores prestaciones. Por último, teniendo en cuenta el bajo volumen de datos que se maneja, se puede concluir con rotundidad, que el mejor sistema de almacenamiento es aquel basado en ficheros.

### 3.4.4. INTERFAZ DE ACCESO

Existen dos posibles soluciones como interfaz de acceso hoy en día. Por un lado, la tradicional aplicación de escritorio en la que el software es instalado en la computadora y se ejecuta como un proceso más del sistema. Y, por otro lado, las aplicaciones web, accesibles desde cualquier ordenador con conexión a internet y navegador instalado. Para tomar una decisión conforme a qué interfaz de acceso escoger, se va a proceder a exponer los pros y los contras de cada una de ellas.

Las **aplicaciones de escritorio** han sido la forma de codificar programas hasta el boom de la web 2.0. La ventaja principal que presenta, se encuentra en la capacidad de hacer un mayor uso de los recursos del sistema. A diferencia de las aplicaciones web, cuyo rendimiento está basado en los medios que le otorgue el sistema operativo al navegador, las aplicaciones de escritorio pueden implementarse atendiendo a su

consumo de CPU, memoria, red, etc. facilitando de esta forma una ejecución más eficiente.

Sin embargo, presenta más inconvenientes que ventajas. Tienen una amplia dependencia con el sistema operativo, de tal forma que es necesario codificar una versión por cada uno de ellos en los que se pretenda que esté presente. Además, ocupan espacio en el disco, por lo que puede llegar a ser un problema si se trata de una aplicación extensa.

Las **aplicaciones web** por su parte, han venido incrementándose a lo largo de los últimos años. Esto es debido entre otras cosas a los grandes avances que han permitido incorporar la lógica necesaria para el funcionamiento en el lado del servidor. Además, la facilidad que otorga la maquetación mediante HTML, junto con la decoración proporcionada por las hojas de estilo en cascada (CSS) y el dinamismo de JavaScript, hace que la implementación de la interfaz gráfica sea muy sencilla y vistosa.

Entre las ventajas de estas últimas, se puede destacar su accesibilidad y que una vez desarrollado es multiplataforma, puesto que únicamente necesitará un navegador y acceso a internet. A esto se le añade, que la única memoria que ocupa en el host es la propia del navegador web. Por último, si es necesario llevar a cabo una actualización, se podrá realizar para todos los usuarios a la vez, tomando las debidas precauciones eso sí.

Aunque si atendemos a las desventajas, tal y como se ha expuesto en la introducción, los ataques de ciberseguridad están en aumento, y el hecho de alojar una aplicación en la web la hace vulnerable a ataques, por lo que habrá que implementar mecanismos que los mitiguen.

Como ha quedado patente, las características que otorgan las aplicaciones web frente a las aplicaciones de escritorio, hacen que sea fácil decantarse por estas a la hora de elegir una interfaz de acceso. Y, aún más, cuando la aplicación que se va a realizar no se espera que necesite un consumo elevado de recursos. Una vez visto, el tipo de interfaz de acceso que se va a implementar, es hora de elegir el *framework* que más se ajuste a la solución a desarrollar.

En la primera sección de este punto, se ha visto que se ha empleado el lenguaje de programación Python 3 para el minado de datos. Partiendo de esta base, sería interesante estudiar como opción un *framework* que emplee dicho lenguaje para la construcción de la interfaz. De esta forma, la sinergia entre ambos módulos de la solución será mayor y se logrará un entorno más homogéneo.

Pueden encontrarse multitud de *frameworks* web en el mercado basados en este lenguaje de programación. Entre ellos cabe destacar Flask, Django o CherryPi. Para la resolución de este trabajo se va a emplear Django, debido a la gran comunidad que le da soporte y a la gran acogida que tiene entre las grandes empresas.

#### 3.4.4.1. DJANGO COMO *FRAMEWORK*

Se trata de un entorno de desarrollo web, *free open source*, basado en Python, que persigue una rápida implementación proporcionando una extensa multitud de herramientas con ese fin. Desde una fácil autenticación de usuarios, permitiendo su gestión de forma segura, hasta la administración del contenido, todo ello por medio de una vista de administrador. Además, proporciona una capa de seguridad frente a ataques de *SQL injection*, *cross-site scripting*, *cross-site request forgery* y *clickjacking*. Punto importante a tener en cuenta, puesto que tal y como se ha mencionado anteriormente, el hecho de llevar a cabo la solución en la red, convierte en un punto crítico la protección de la misma. A esto se le añade su escalabilidad a la hora de gestionar el tráfico y su versatilidad. [19]

Para esta lograr la funcionalidad por la que se caracteriza, la rapidez de implementación, posee una estructura muy estudiada. Primeramente, es necesario generar un proyecto Django. Éste, constituirá un contenedor donde se alojarán diferentes aplicaciones referentes al mismo. Las aplicaciones son entendidas por Django como distintos módulos de la solución que pueden ser exportables e importables, permitiendo de esta forma, seguir el principio que la misma organización definió: DRY (***D**on't **R**epeat **Y**ourself*) ahorrando así, esfuerzos de codificación.

Ya se ha definido como queda estructurado el proyecto a nivel de aplicaciones, sin embargo, no se ha hecho referencia todavía a como se encuentran configuradas éstas. Pues bien, cada una de ellas ha de poseer, como mínimo, los siguientes ficheros para su correcto funcionamiento:

- **urls.py:** en este fichero se detallarán todas la *urls* relativas por las que se encontrarán las vistas al hacer una petición HTTP(s)
- **views.py:** es en este archivo donde se codificará la lógica que se desee aplicar en la parte del servidor. Además, se definirán también las plantillas a las que se redirigirán las peticiones.

Además de estos, el directorio correspondiente al proyecto, contendrá el fichero "***settings.py***", en el cual se indicarán todas aquellas configuraciones necesarias para el buen desempeño del mismo.

En cuanto a la transferencia de datos Servidor-Cliente, es capaz de mandar los datos necesarios a las plantillas HTML en el contexto del *response*. De esta forma, las vistas de las páginas podrán generarse de manera dinámica, en función, por ejemplo, del número de elementos que disponga una lista pasada por contexto. Proporciona una serie de mecanismos que permiten implementar bloques de condiciones, así como bucles de iteración, factores que resultan realmente útiles.

#### 3.4.4.2. API REST

Es una de las tecnologías más empleadas desde que surgió en el 2000 y revolucionó el mundo del desarrollo de software. Permite el acceso o manipulación de datos de manera sencilla mediante el protocolo de comunicación HTTP. Cada petición contiene la información necesaria para ejecutarla, por lo que cliente y servidor no tienen por qué conocer el estado previo para llevarla a cabo. [20]

Es una manera sencilla de implementar un acceso al sistema para desarrolladores que deseen hacer uso de los datos del mismo. Además, mejora la portabilidad a otros tipos de plataformas permitiendo, de esta manera, una mayor independencia del lenguaje.

Su uso básico es que el desarrollador implemente un código que, mediante una consulta HTTP, con unas determinadas cabeceras, especificadas según se haya implementado la API, pueda ejecutar determinadas acciones en el servidor del sistema y obtener una respuesta con los resultados. Estos resultados pueden ser devueltos en muchos formatos, pero el más común es el formato JSON, debido a su sencillez y su potencial.

## 3.5. SÍNTESIS

Para cerrar el punto de análisis, se va a sintetizar todos aquellos aspectos referentes a las tecnologías que se han tratado en él.

La extracción de los datos se hará por medio de *web scrapers*, uno para cada fuente de datos, ya que tal y como se ha podido ver en su análisis, presentan bastantes diferencias. Se ha decidido codificarlos, frente a otras soluciones presentes en el mercado, ya que facilitará la automatización de la tarea.

El almacenamiento de los datos se hará por medio de sistemas de ficheros. Aunque las ventajas que presentan las bases de datos, tanto relacionales como no relacionales, son muy notables, la naturaleza de nuestro sistema no va a permitir al usuario hacer otras peticiones que no sean de consulta. De este modo, la funcionalidad de estas tecnologías no quedaría totalmente cubierta al no tener que procesar consultas de edición de datos. Los sistemas de ficheros, por su parte, pueden implementarse atendiendo a las necesidades que requieran las aplicaciones, además no requieren de espacio de memoria adicional al únicamente empleado por los archivos.

Por último, debido a la tendencia del desarrollo software, la interfaz de acceso ha de ser vía aplicación web con su respectiva API REST. Las ventajas que presenta son bastante superiores a la de las aplicaciones de escritorio. La implementación irá de la mano del *framework* Django, codificado en Python, que facilita la gestión de usuario, así como los aspectos de seguridad.

# 4. DISEÑO E IMPLEMENTACIÓN DE LA SOLUCIÓN

Una vez se ha llevado a cabo el análisis de todas las tecnologías que conformarán el proyecto, es hora de empezar con el diseño y la implementación de los distintos módulos y aspectos de la solución.

En este capítulo, se describirán aquellas decisiones que se hayan tenido que tomar conforme al diseño del producto. También se expondrá como estas se han llevado a la práctica.

Se comenzará comentando todos aquellos aspectos relevantes a la creación del sistema de ficheros y su contenido. Más tarde, se describirá el diseño de los distintos *web scrapers*, tanto de ENISA como de SEEDLabs junto a la implementación. Luego, se presentará el análisis de los términos de los documentos obtenidos previamente en el paso anterior. Estos dos últimos puntos mantienen una fuerte relación con el sistema de ficheros, por lo que también se referenciará al mismo en bastantes ocasiones. Más tarde, se entrará de lleno en los aspectos de autenticación de usuarios, tanto en la interfaz web, como en la API REST. La búsqueda básica y la búsqueda avanzada serán tratadas más adelante, para terminar con el diseño e implementación de la API REST y las consideraciones de seguridad.

## 4.1. SISTEMA DE FICHEROS

El sistema de almacenamiento, es uno de los puntos clave en el diseño e implementación de la solución. Realizarlo de la manera correcta, permitirá aumentar la eficiencia de la aplicación y los recursos que consuma. Para ello, es necesario conocer qué es lo que se va a guardar en ellos.

Por un lado, deberá contener un archivo en el que se reflejen todos los metadatos de cada training. Por otro lado, deberá disponer de varios ficheros que almacenen todos los términos para cada ejercicio. Finalmente, estos archivos deberán estar organizados por directorios con el fin de hacerlos identificables.

A continuación, se procederá a la descripción de todos estos aspectos. Sin embargo, es necesario recalcar que, el almacenamiento de datos a nivel de implementación, guarda una amplia relación con los scripts de minado de datos y de análisis de palabras, tal y como se puede prever, puesto que son estos los que obtienen los datos para después guardarlos. Es por esto, que el detalle de como se ha implementado la recogida de datos para su posterior volcado en los ficheros, queda retratado en los puntos 4.2 y 4.3.

### 4.1.1. DIRECTORIOS

Es el punto principal del sistema de ficheros. Regirán la organización de los datos y su complejidad o sencillez determinarán el nivel de dificultad de acceso a los mismos.

Para el desarrollo de la solución se ha decidido que, tal y como se ha mencionado en el punto de análisis, todos los archivos pertenecientes a cada training se analicen en busca de los términos que poseen, por lo que será necesario almacenarlos durante un mínimo periodo de tiempo. Para cumplir con esto, manteniendo una cierta organización, se llevará a cabo un árbol de directorios en el que de la raíz partan directorios con el nombre de la fuente de datos a la que van a corresponder, junto con la palabra "*Files*". De esta manera, dentro de cada una de estas carpetas, se pueden generar a su vez una lista de carpetas cuyo nombre se corresponda al propio del ejercicio. Para finalmente, dentro de cada uno de estos directorios, almacenar todo aquel fichero que se corresponda con él.

Los ficheros que guardarán relación con el training, no sólo serán aquellos necesarios para el entendimiento o realización del mismo, sino que también, tal y como veremos en el punto 4.1.3, el fichero generado con los términos que se hayan extraído de los archivos mencionados.

Para terminar este subapartado, se puede consultar el **ANEXO II** con la estructura que dispondría el árbol de directorios, una vez se hayan minado los datos y analizado sus documentos. Cabe destacar, que la implementación dicho árbol, se llevará a cabo de manera automática en los scripts destinados al *web scraping*.

### 4.1.2. JSON

JSON (JavaScript Object Notation) es un formato ligero de intercambio de datos. Es fácil para los humanos leer y escribir. Así como para las máquinas analizar y generar. Se basa en el lenguaje de programación de JavaScript y es altamente compatible con Python al ser bastante similar a sus diccionarios. [21]

Puede construirse sobre 2 estructuras, una colección de clave: valor y una lista ordenada de valores siempre separados por comas. Los valores pueden variar entre cadenas de texto, números, otro objeto JSON, otra lista ordenada y booleanos. En las figuras inferiores pueden observarse las arquitecturas de cada una de las estructuras y valores.

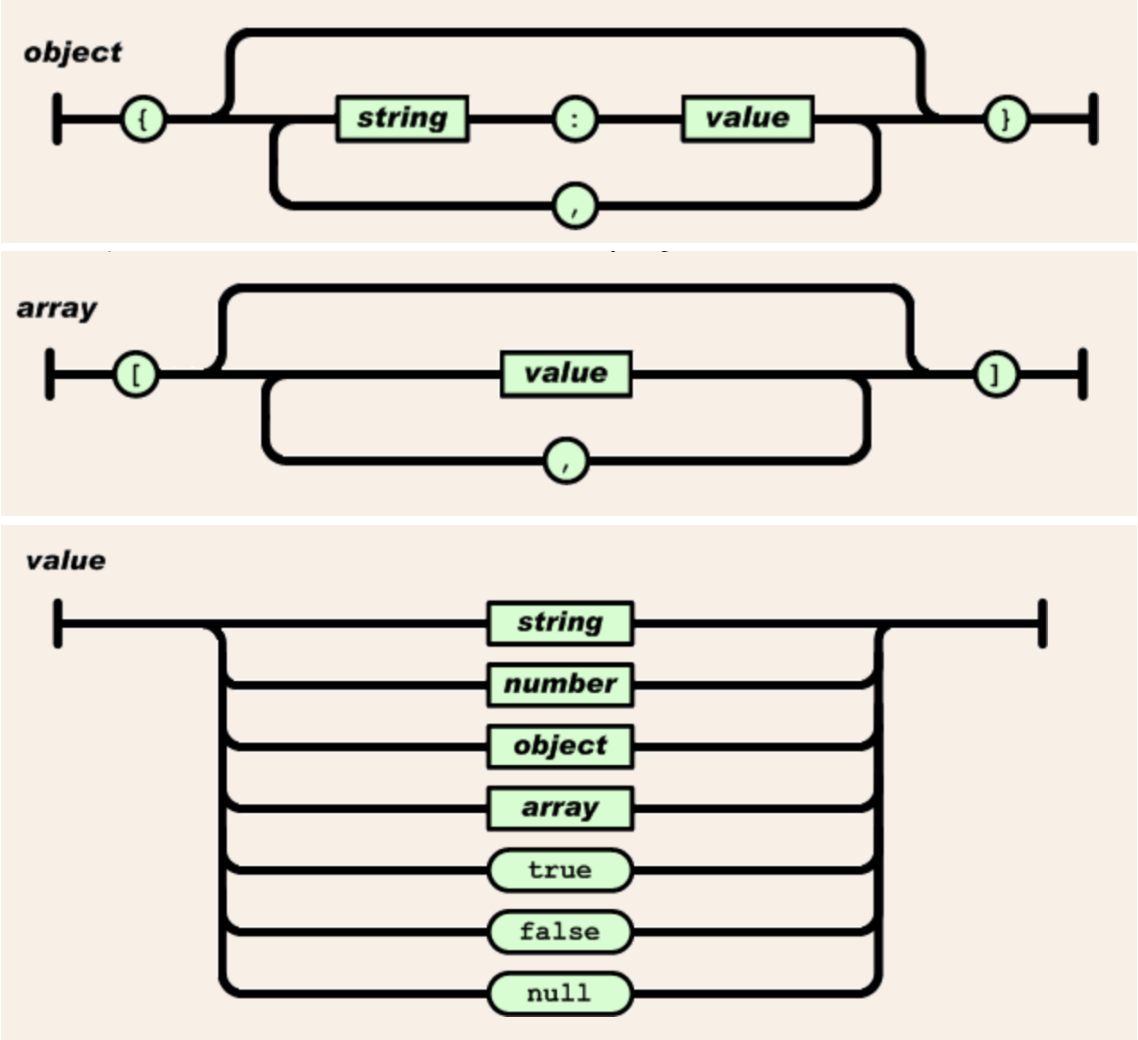



*Ilustración 9 - Arquitecturas de las estructuras JSON [21]*

Estas propiedades hacen de él, el perfecto aliado para llevar a cabo el almacenamiento de metadatos. En la solución, la estructura que se ha decidido que tenga conforme a los datos elegidos a almacenar, es la siguiente:

1. Un objeto "*resources*" que englobe en una lista ordenada con objetos de todos los ejercicios de las distintas fuentes de origen de datos.
2. Dentro de la lista, una secuencia ordenada de objetos que hacen referencia individualmente a cada training. De tal forma, que cada objeto almacena un conjunto de claves: valores, correspondientes a los datos extraídos.
3. En alguna de esas claves, se almacenará una lista ordenada de valores, como puede ser el caso de *files* o *urls*.

En la siguiente imagen puede observarse parte de la composición del fichero JSON, de nombre "***data.json***" obtenido mediante la ejecución de los *web scrapers*. Con este adjunto, se pretende facilitar la comprensión de la estructura al lector.

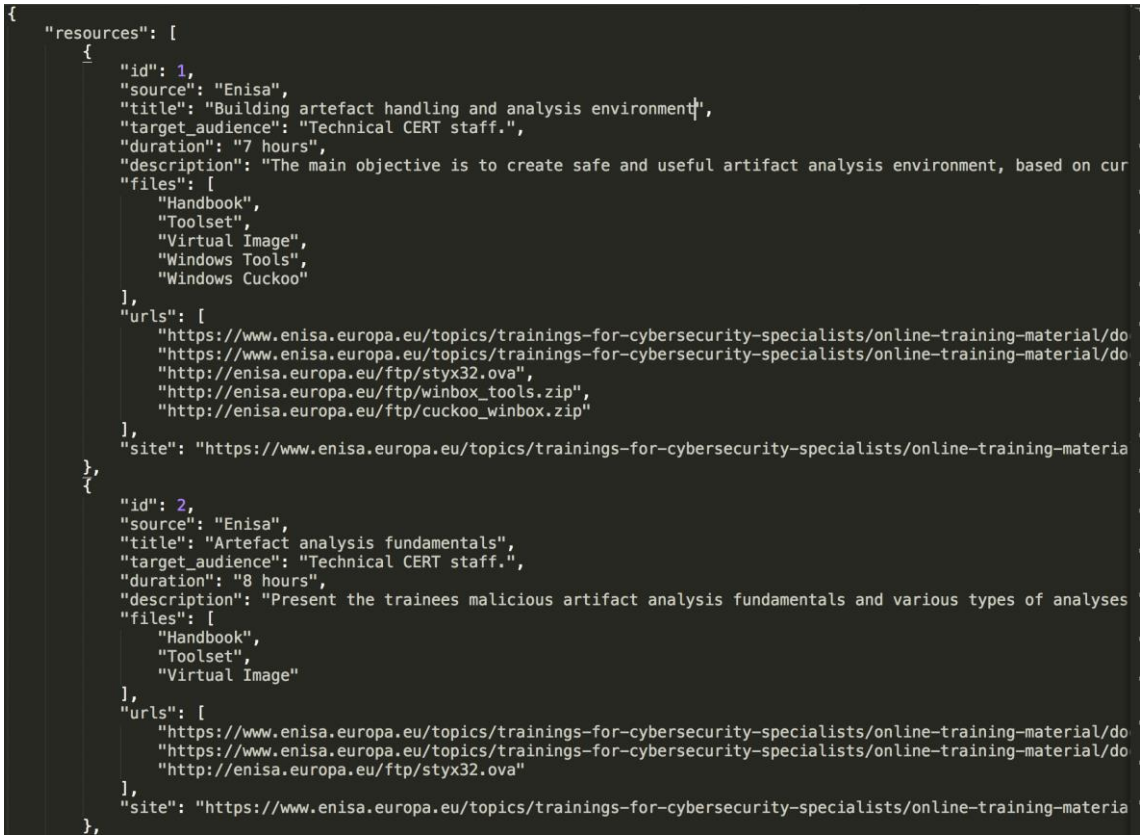



*Ilustración 10 - Ejemplo del fichero generado por los web scrapers*

Por último, dicho archivo se almacenará en la raíz del árbol de directorios explicado en el anterior subpunto. De esta forma, el acceso al mismo no quedará sepultado bajo un cúmulo de carpetas, facilitándolo y mejorando los tiempos de respuesta.

### 4.1.3. FICHERO DE TÉRMINOS

Tal y como se ha podido leer en la sección de análisis, hay que realizar un procesamiento de los ficheros que componen los ejercicios de cada fuente, en busca de términos de relevancia, por las que el usuario pueda encontrar un material en concreto. Este hecho, requiere almacenar dichas palabras de alguna manera. Para ello, se ha pensado volcar todas las palabras obtenidas del análisis en un fichero de texto que las contenga y permita al software, su búsqueda, para arrojar los resultados exitosos.

De esta manera, por cada ejercicio alojado en las distintas fuentes, se generará un fichero de texto, llamado "***mainwordfile.txt***", que se almacenará en la carpeta cuyo

título se corresponda con el nombre del training. Más adelante, en el punto 4.3 se verá la implementación seguida para lograr este hecho.

## *4.2. WEB SCRAPER*

Como se ha comentado anteriormente en el punto de análisis de la solución, el minado de datos se llevará a cabo por medio de un software que se conecte a las *urls* dadas y mediante la conversión del código HTML, obtenga la información que hemos definido en el análisis de las fuentes de datos del apartado anterior. De esta manera, se podrá proceder a su almacenamiento en el archivo JSON, que será almacenado en el sistema de ficheros, tal y como se explicó en el punto anterior.

Como la implementación de los dos minadores de datos se llevará a cabo de manera independiente, es necesario establecer un orden de escritura en el fichero. Por ello, el primero en ejecutarse, y por tanto en escribir, será el correspondiente a ENISA, para más tarde, el referente a SEEDLabs, lea el fichero y le añada la información que haya obtenido de su fuente.

Una vez explicada en líneas generales, los aspectos comunes a ambos web scrapers, se procederá a detallar cada uno de ellos.

### 4.2.1. ENISA

El aspecto más importante a tener en cuenta para el diseño de este extractor de datos, es que los ejercicios de ENISA no disponen de una vista individual. Si no que los datos de todos los trainings de una categoría dada, se muestran en el mismo HTML. Esto supone que, en lugar de ir actualizando el fichero a medida que se obtienen datos de un ejercicio, habrá que recopilar todos los datos, para posteriormente agruparlos por su training correspondiente y escribirlos en el JSON. A continuación, se expondrá una tabla en la que se puede apreciar las etiquetas HTML correspondientes a cada uno de los datos de interés para cada ejercicio.

| Metadato | Etiqueta HTML |
|---|---|
| **title** | **<h2>** |
| **description** | <table><tbody><tr[3]><th>**<p>** |
| **target_audience** | <table><tbody><tr[2]><th[1]>**<p>** |

| | |
|---|---|
| **duration** | <table><tbody><tr²><th²>**<p>** |
| **files** | <table><tbody><tr²><th³>**<p>** |
| **urls** | <table><tbody><tr²><th³><p>**<a[href]>** |

*Tabla 1 - Posición HTML de los metadatos ENISA*

**NOTA 1: Los superíndices hacen referencia a la posición de la etiqueta dentro de la tabla HTML. De tal forma que el superíndice <tr¹><th¹> hace referencia a la primera fila y primera columna.**
**NOTA 2: Los corchetes indican que donde se encuentra el dato es en ese mismo atributo de la etiqueta HTML.**

A estos campos obtenidos, es necesario añadirles otros que pese a que no están explícitamente indicados son de gran interés:

- **id**: para facilitar la identificación del ejercicio.
- ***source*:** indica la fuente de origen del material.
- ***site***: la url de la categoría del training con el fin de poder referenciarlo más adelante en la construcción de la web.

El motivo por el cual se va a decidido almacenar las urls de los ficheros es porque se va a proceder a descargarlos, con vistas a poder analizarlos en busca de palabras clave. De esta manera, tenemos dos funcionalidades para el script. La primera de ellas, extraer los datos definidos, mientras que la segunda descargar los ficheros y generar el correspondiente sistema de directorios. Para ello se va a hacer uso de los módulos mencionados durante el análisis de la solución: *"wget"* [22] para las descargas y *"Beautifulsoup"* [23] para el minado de datos. Además de estos, también se emplearán el módulo *"os"* para la gestión de directorios, el módulo *"requests"* para realizar peticiones HTTP y por último el módulo *"json"* para el volcado de datos.

Una vez visto el diseño, se detallará la implementación del mismo. Dicho lo cual, se comenzará describiendo el flujo de conexiones. En el siguiente fragmento de código se podrá observar cómo se lanza una petición GET HTTP a la *url* que se menciona en la sección de fuentes de datos, del punto análisis de la solución. Posteriormente, mediante el uso de la librería de *"Beautifulsoup"* se convierte el contenido de la respuesta a HTML.

```
url = requests.get("https://www.enisa.europa.eu/topics/trainings-for-cybersecurity-specialists/online-training-material")
response = BeautifulSoup(url.content,"html.parser")
```

*Ilustración 11 - Código de consulta HTTP y conversión de la misma*

Hecho esto, ya se tiene el árbol HTML, con todas las etiquetas, por lo que habrá que proceder a obtener aquellas que sean de utilidad. Recordando lo mencionado en el apartado correspondiente del análisis. Esta vista de la web, segmentaba los trainings en 4 categorías, dentro de las cuales, se encontraban los detalles de cada uno de los ejercicios. Es por esto que, primero se deberá obtener el enlace a todas estas categorías para más tarde repetir el proceso descrito por cada uno de las *urls* obtenidas.

```
21  # Obtiene todos los links de las diferentes clasificaciones de trainings
22  for link in response.findAll('tr'):
23      if link.td.find('a') != None:
24          traininglinks.append(link.td.a['href'])
25  # Conecta con cada uno de los links y obtiene los datos deseados.
26  for urltraining in traininglinks:
27      page = requests.get(urltraining)
28      print("CONECTANDO A: {0} ...".format(urltraining))
29      responsetr = BeautifulSoup(page.content,"html.parser")
30      for title in responsetr.find_all('h2'):
31          if title.text != None:
32              titles.append((title.text.replace("\u00a0"," "),urltraining))
33
34      for content in responsetr.find_all('table'):
35          if content.findAll('p') != None:
36              contenttr = content.find_all('p')
37              information = []
38              resources = []
39              for p in contenttr:
40                  information.append(p.text.replace("\u00a0"," "))
41              resourcestr = content.find_all('a')
42              for a in resourcestr:
43                  resources.append(a['href'])
44              informationdivided.append((information,resources))
```

*Ilustración 12 - Código para la obtención de los metadatos*

En la imagen superior (Ilustración 12) puede apreciarse como se obtienen las *urls* de las diferentes categorías para más tarde hacer un bucle sobre ellas y recoger aquellos datos que son de interés.

Se puede observar como dentro de cada iteración, primeramente, se guarda en una lista de Python, todos los nombres de los ejercicios pertenecientes a la categoría de la que se ha recuperado el contenido, así como la *url* de la categoría. Más tarde, se itera por todas las tablas del contenido, para de esta forma obtener los datos relevantes mostrados en la tabla del comienzo del apartado (Tabla 1).

Tal y como se puede ver, en las líneas 39-40 (Ilustración 12), los datos referentes a la audiencia, la duración, la descripción y los archivos son almacenados en una lista llamada "*information*". Más tarde, en las líneas 42-43, se procederá a iterar por cada uno de los hipervínculos, para ir añadiéndolos a una lista llamada "*resources*", donde se almacenarán todas las *urls* destinadas a la posterior descarga de los ficheros.

Por último, ambas listas, *information* y *resources*, serán embebidas en una dupla para más tarde añadirlas a la lista de "*informationdivided*", donde se almacenará toda la información relevante a los trainings de cada categoría.

A continuación, una vez expuesto el diseño e implementación del minado de datos, se procederá a hacer lo propio con el volcado de datos al fichero.

```
45  count = 0
46  resources.reverse()
47  for title_url in titles:
48      folder = FILE_PATH
49      folder = os.path.join(folder, title_url[0])
50      if not os.path.exists(folder):
51          os.makedirs(folder)
52
53      data['resources'].append({
54          'id':count,
55          'source': 'Enisa',
56          'title': title_url[0],
57          'target_audience': informationdivided[count][0][0],
58          'duration': informationdivided[count][0][1],
59          'description': informationdivided[count][0][-1],
60          'files': informationdivided[count][0][2:-1],
61          'urls': informationdivided[count][1],
62          'site': title_url[1]
63          })
64
65
66      with open('data.json', 'w') as outfile:
67          json.dump(data,outfile, indent = 4)
68
69      for file in informationdivided[count][1]:
70          dwnfile = wget.download(file, out = folder)
71
72      count+=1
```

*Ilustración 13 - Código que genera el fichero JSON y el árbol de directorios*

En la figura (Ilustración 13), puede observarse como se itera por cada uno de los elementos de la lista que contenía los títulos y la *url* de la categoría. Cabe destacar que, las líneas 48-51 son referentes al sistema de ficheros, en ellas se accede al directorio cuyo nombre coincide con el título del training y en caso de no existir, crearía la carpeta. En las líneas 53-63 se procede a convertir el correspondiente diccionario de Python a formato JSON, para más tarde en las líneas 66-67 escribirlo en el fichero. Finalmente, las líneas 69 y 70 proceden a iterar por todos los ficheros del training y a descargarlos en sus directorios correspondientes.

### 4.2.2. SEEDLABS

La principal diferencia con respecto a la otra fuente de datos es que, en este caso, los materiales sí disponen de una vista propia en la web. Este hecho, junto a la distinta estructura del código HTML y la diferencia de los campos de interés, hará que la implementación siga una lógica distinta, tal y como se verá a lo largo de la sección. Antes de proceder a describir el desarrollo del mismo, se va a proceder a enumerar aquellas propiedades que se deseen almacenar, así como su lugar en el árbol HTML.

En el análisis de las fuentes de datos se ha determinado que ciertos datos se obtendrían de las distintas vistas de la web. De tal forma que, los campos de la siguiente tabla se obtendrán de la página en la que se listan todos los trainings de la categoría (Ilustración 7).

| Metadato | Etiqueta HTML |
|---|---|
| **title** | <ul><li><a>**<h3>** |
| **description** | <ul><li><a>**<p>** |
| **type** | <ul><li><a><h3>**<img [src]>** |
| **difficulty** | <ul><li><a>**<h3[class]>** |

*Tabla 2 - Posición HTML de los metadatos en la vista de categoría de SEEDLabs*

Cabe destacar que, como la dificultad viene indicada mediante una imagen, habrá que realizar un procesamiento de la *url* de su atributo *src*, en busca de las tres clasificaciones que hace la fuente: ataque, exploración o implementación. Además, en cuanto al campo "*type*", al ser un atributo clase, no todos los ejercicios lo poseen, por lo que en aquellos que no se encuentre se fijará un valor predeterminado: *"Not defined".*

La siguiente vista de la que se obtendrán más datos, será aquella propia del material en sí (Ilustración 8). En ella, tal y como se ha podido observar de nuevo en la sección de análisis de las fuentes de datos, se obtendrán los siguientes campos:

| Metadato | Etiqueta HTML |
|---|---|
| **description (más detallada)** | **<p>** |
| **files** | <h3>**<a>** |
| **urls** | <h3>**<a[href]>** |

| | |
|---|---|
| **duration** | buscar “week” en el texto |
| **files (needed)** | <ul[3]><li>**<a>** |
| **urls (needed)** | <ul[3]><li>**<a[href]>** |

*Tabla 3 - Posición HTML de los metadatos de la vista de ejercicio de SEEDLabs*

**NOTA: Los superíndices en este caso, hacen referencia a la posición en la lista desordenada de HTML. De tal forma que, el 3 índica que es el tercer punto de la lista.**

De nuevo, al igual que en el caso anterior, es necesario añadirle ciertos campos que van implícitos en la naturaleza del material. Como son:

- **id:** para facilitar la identificación del ejercicio.
- ***source*:** indica la fuente de origen del material.
- ***site*:** la *url* de la vista del training con el fin de poder referenciarlo más adelante en la construcción de la web.

En este caso, el motivo por el cual se ha decidido guardar las urls de los ficheros es el mismo que en el anterior. Sin embargo, existe una peculiaridad con este tipo de dato. Si se analiza el atributo en el que están contenidas, se puede observar como hace uso de *url* relativas, en lugar de absolutas. Este hecho, obligará a emplear mecanismos que sean capaces de recomponer la *url* de manera absoluta, con el fin de más tarde, poder ejecutar consultas exitosas, que nos den como resultado la descarga del material o la visualización del mismo. Al igual que antes, los módulos empleados serán *“wget”* [22] para las descargas y *“Beautifulsoup”* [23] para el minado de datos, *“os”* para la gestión de directorios, *“requests”* para realizar peticiones http y *“json”* para el volcado de datos.

Estudiado el diseño de este sitio web, es hora de proceder a la implementación del script que recogerá los datos y los guardará en el fichero JSON existente. Para ello, lo primero que se llevará a cabo será cargar los datos existentes de dicho fichero en un diccionario de Python y obtener el último id. Para más tarde, realizar la consulta a la url indicada en la sección de análisis de la solución y convertir la respuesta a HTML, tal y como se puede observar en la siguiente figura.

```
11  with open("data.json",'r') as datafile:
12                  data = json.load(datafile)
13
14  lastid = data['resources'][-1]['id']
15
16  urlrsc = "http://www.cis.syr.edu/~wedu/seed/Labs_16.04/"
17  url = requests.get(urlrsc)
18  response = BeautifulSoup(url.content,"html.parser")
```

*Ilustración 14 - Código que lee datos, realiza consulta HTTP y la convierte.*

Una vez se posee el contenido convertido, se procede a buscar todas aquellas categorías en las que se clasifican los ejercicios. Para ello, se buscan todas aquellas etiquetas  cuya clase sea "*one_third*" puesto que en el atributo "*href*" de su nodo hijo <a> vendrá la url de las mismas. Todos estos enlaces, serán almacenados en una lista por la que se iterará para ir accediendo a cada una de las vistas de las categorías y obtener su contenido. Todo esto, puede observarse traducido a código en la siguiente ilustración.

```
29  for link in response.findAll("div",{"class":"one_third"}):
30      traininglinks.append(link.a['href'])
31
32  for urltraining in traininglinks:
33      urlres = urlrsc + urltraining
34      url = requests.get(urlres)
35      print("\nConnecting to: {0}".format(urlres))
36      response = BeautifulSoup(url.content,"html.parser")
```

*Ilustración 15 - Código que itera por los distintos enlaces de categorías*

Una vez se ha hecho esto, ya se posee el contenido convertido de la vista de la categoría. En ella, han de recogerse los datos relativos a cada training presente en la primera tabla de esta sección (Tabla 2). Para acometer esto, se iterará por todas aquellas etiquetas <li> (línea 38 – Ilustración 16), obteniendo de sus nodos hijos la información que se desea, no sin antes hacer las comprobaciones pertinentes que aporten robustez. En el caso de la dificultad (variable *diff* - línea 42 – Ilustración 16) puede observarse como en caso de que no exista el atributo *class*, se guardará como no definida. Con el tipo de ejercicio, al obtenerse según la palabra que esté presente en el atributo *src* de la imagen, es necesario efectuar esa búsqueda de palabras (líneas 48-53 – Ilustración 16). Por último, la descripción breve se obtiene del texto del elemento <p>. El fragmento de código descrito abajo, representa cómo se ha llevado a cabo lo explicado en este párrafo.

```
38      for linkrsc in response.findAll('li'):
39          if linkrsc.h3 != None:
40              title = linkrsc.h3.text
41              img = linkrsc.h3.img['src']
42              diff = linkrsc.h3.get('class')
43              if diff != None:
44                  diff = diff[0]
45              else:
46                  diff = "Not defined"
47
48              if "attack" in img:
49                  typetr = "attack"
50              elif "exploration" in img:
51                  typetr = "exploration"
52              else:
53                  typetr = "implementation"
54          if linkrsc.p != None:
55              description = linkrsc.p.text
```

*Ilustración 16 - Código que obtiene parte de los metadatos*

Ya que se han obtenido todos los datos de la vista por clasificación, es hora de adentrarse en la url que da acceso a la vista individual del training. En la siguiente figura podrá observarse como se ha llevado a cabo este acceso y el proceso de minado del resto de datos.

```
57      if urlrsc in linkrsc.a['href']:
58          print("\nConnecting to: {0}".format(linkrsc.a['href']))
59          urltr = linkrsc.a['href']
60          urltrequest = requests.get(linkrsc.a['href'])
61          root = linkrsc.a['href']
62      elif "drive" in linkrsc.a['href']:
63          pass
64      elif ".pdf" in linkrsc.a['href'] or ".zip" in linkrsc.a['href']:
65          pass
66      else:
67          urltr = urlres + linkrsc.a['href']
68          root = urltr
69          print("\nConnecting to: {0}".format(urltr))
70          urltrequest = requests.get(urltr)
71
72      response = BeautifulSoup(urltrequest.content,"html.parser")
73      files = []
74      dwnfiles = []
75      information = []
76      urlfiles = []
77      for p in response.findAll('p'):
78          description += "\n" + p.text
79      for h3 in response.findAll('h3'):
80          for res in h3.findAll('a'):
81              files.append(res.text)
82              if urlrsc in res['href']:
83                  urlfiles.append(res['href'])
84              else:
85                  finalurl = urltr + res['href']
86                  urlfiles.append(finalurl)
87      for li in response.findAll('li'):
88          if li.a != None and "files/" in li.a['href']:
89              files.append(li.a.text)
90              dwnfiles.append(li.a['href'])
91          elif li.a !=  None and ".py" in li.a['href']:
92              files.append(li.a.text)
93              dwnfiles.append(li.a['href'])
94
95      rootlist = root.split('/')
96      rootpath = root + rootlist[-2]
97      rootpdf = rootpath + '.pdf'
98      for dur in response.find(string = re.compile("week")):
99          information.append(dur)
100
101     for file in dwnfiles:
102         filepath = root + file
103         if requests.get(filepath).status_code == 200:
104             urlfiles.append(filepath)
```

*Ilustración 17 - Código que mina el resto de metadatos*

Tal y como se ha comentado anteriormente, existe un inconveniente a la hora de tratar las *urls* relativas. Es en este punto, donde sin dudas se han encontrado más problemas a la hora de la implementación. Por un lado, ha hecho falta comprobar si la *url* era absoluta o relativa, tal y como se puede ver en la línea 57 (Ilustración 17). La lógica implementada consiste en estudiar si el atributo *href* contiene la *url* de la vista del *training*. Si es así, se tratará de una *url* absoluta y no habrá que hacer operaciones sobre ella. Si no, es así, se han tenido que filtrar todas aquellas *urls*, que contuvieran "drive", ".pdf" o ".zip", tal y como se aprecia en las líneas 62-65 (Ilustración 17). Si la *url*, no cumple ninguna de las condiciones de las líneas 57, 62 y 64, entonces significará que es relativa y habrá que convertirla en absoluta. Este proceso es el que se lleva a cabo en las líneas 66-70 (Ilustración 17).

Tras tener la *url* absoluta, se realiza la *request* y se convierte la *response* correspondiente (línea 72 – Ilustración 17). Con el contenido preparado, lo único que queda es obtener aquellos datos que son de interés y almacenarlos en listas para su posterior almacenado en el fichero JSON.

Por último, toda la información es añadida al diccionario que se cargó al principio, el cual poseía ya los datos extraídos de ENISA. Más tarde, se crea el sistema de directorios (líneas 119-122 – Ilustración 18) para proceder a la descarga de los archivos correspondientes a cada training (líneas 124-128 – Ilustración 18). Finalmente, se convierte el diccionario a formato JSON y se guarda en el mismo fichero de datos que constituirá el repositorio de metadatos.

```
106            data['resources'].append({
107                'id':lastid+1,
108                'source': 'SEEDLabs',
109                'title': title,
110                'type' : typetr,
111                'difficulty': diff,
112                'duration': information[0] + " week(s)",
113                'description': description,
114                'files': files,
115                'urls': urlfiles,
116                'site':urltr
117                })
118            lastid += 1
119            folder = FILE_PATH
120            folder = os.path.join(folder,title.replace("/","-"))
121            if not os.path.exists(folder):
122                os.makedirs(folder)
123
124            download = wget.download(rootpdf,out = folder)
125            for file in dwnfiles:
126                filepath= root + file
127                if requests.get(filepath).status_code == 200:
128                    downloadfile = wget.download(filepath, out = folder)
129
130    with open("data.json",'w') as outfile:
131        json.dump(data,outfile,indent = 4)
```

*Ilustración 18 - Código que inserta los nuevos metadatos y descarga los contenidos*

## 4.3. ANÁLISIS DE TÉRMINOS

El análisis de las palabras contenidas en los ficheros obtenidos previamente, mediante la descarga realizada por los scripts de minado de datos, es el último paso del módulo de la solución que consiste en la elaboración del *dataset*.

Obviamente, al hacer uso de los datos obtenidos por los scripts mencionados anteriormente, será ejecutado posteriormente a ellos, una vez el sistema de ficheros esté totalmente creado y sea accesible.

Este script, al igual que los anteriores hará uso de Python 3, concretamente de los siguientes módulos:

- **os:** para navegar por el sistema de directorios.
- **shutil:** para mover los ficheros generados a su carpeta correspondiente.
- **PyPDF2:** para convertir los archivos PDF's en texto interpretable por Python.
- **textract:** convierte imágenes de los PDF's en texto.
- **nltk:** para llevar acabo el análisis de palabras clave.

En cuanto al **diseño de la solución**, se ha decidido que una vez se hayan analizado todos los documentos PDF propios de cada *training*, se generará un fichero de texto único a todos ellos, que contenga todas las palabras resultantes. Antes de pasar al análisis de términos del resto de ejercicios, se moverá el fichero resultante al directorio que le corresponda, según el nombre del mismo y la fuente de datos de la que provenga. De esta manera, cuando se implemente la búsqueda, tal y como se verá en el punto 4.4.3.3 y 4.4.3.4, podrá verse fácilmente que palabras clave contiene cada training, identificándolos por el nombre y haciendo una búsqueda en el fichero de texto, que mantendrá el mismo nombre para todos: ***"mainwordfile.txt"***.

Respecto a la implementación, cabe mencionar que la lógica es exactamente la misma para cualquiera que sea la fuente de la que provengan los ficheros a analizar. De hecho, se cumple de esta manera, el objetivo de crear una infraestructura que soporte el añadido de nuevas fuentes de datos.

Lo primero que realiza el script es situarse en la raíz del sistema de directorios creado. Ahí donde se alojan los *web scrapers*, el fichero JSON con los metadatos y el software al que nos referimos. Desde ahí, mirará el listado de ficheros y directorios e iterará por todos ellos. En el caso de que alguno de ellos, contenga en su nombre la palabra "*Files*", significará que dentro aloja a su vez el árbol de directorios correspondiente a los títulos

de cada training, tal y como se ha dejado patente en el punto 4.1.1. De tal forma que, entrará dentro del mismo y procederá a recorrer el árbol, analizando todos los ficheros PDF que encuentre por el camino. Finalmente, generará el fichero de texto con todas aquellas palabras clave que haya procesado.

Para el desarrollo del software se ha seguido una guía de *Medium* adaptándola a la arquitectura de la aplicación. [24]

## 4.4. INTERFAZ DE ACCESO

La interfaz de acceso es un punto vital a la hora de diseñar la solución. Esto se debe a que supone el primer punto de contacto del usuario con la aplicación. Su diseño ha de ser intuitivo y sencillo, puesto que uno complejo dificultaría el propósito por el cual se accede al sistema.

Otro de los aspectos con los que el usuario tiene mayor relación es con el uso de las cuentas en el sistema. Y es que, para proveer de mecanismos de autenticación eficientes, se ha determinado que el sistema va a tener que desarrollar mecanismos de gestión de cuentas. Es por esto que, hay que tratar de realizarlo de la manera más cómoda posible para la persona que haga uso de la solución.

A lo largo del siguiente subapartado, se detallará la estructura que se ha ideado para el proyecto de Django. También, se expondrán las decisiones de diseño e implementación tomadas para tratar la administración de usuarios, la interfaz web y la API REST.

### 4.4.1. ESTRUCTURA DEL PROYECTO DJANGO

Como se ha mencionado anteriormente, Django posee la característica habilidad de poder crear aplicaciones dentro de un mismo proyecto, que pueden ser exportables. Como uno de los objetivos de este trabajo de final de grado es la granularidad. Se ha decidido dividir el proyecto en dos aplicaciones: la referente a la interfaz web y aquella que guarda relación con la API REST. En la siguiente figura se puede observar su estructura.

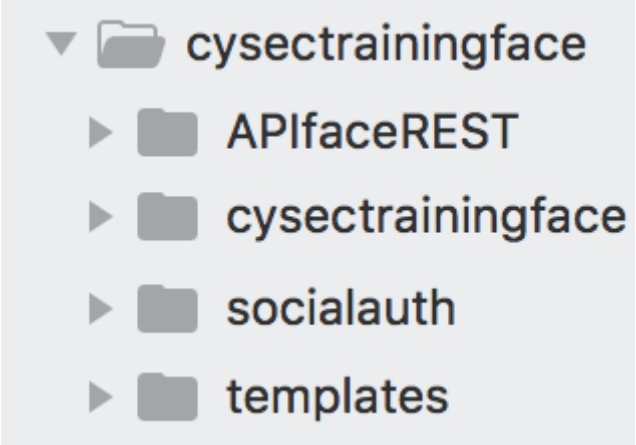



*Ilustración 19 - Árbol de directorios del proyecto Django*

Puede observarse como del nodo raíz del proyecto "*cysectrainingface*", cuelgan cuatro directorios, de los cuales, los tres primeros se corresponden a aplicaciones y el último a plantillas HTML. Como se podrá intuir, el directorio "*APIfaceREST*" contendrá lo necesario para permitir la funcionalidad de la API REST que se estudiará en el punto 4.4.3. La llamada "*socialauth"*, gestionará lo relevante a la interfaz web. Por último, "cysectrainingface", generada al crear el proyecto Django, poseerá la configuración relativa a todo el proyecto.

Este mismo estará contenido en la raíz de nuestro sistema de ficheros, de tal forma que, cuando una de estas aplicaciones, necesite acceder a los datos del repositorio, únicamente tendrá que abstraerse un nivel para encontrar el fichero "*data.json*" donde estarán contenidos todos aquellos metadatos recogidos previamente por los *web scrapers*. Así como, todos aquellos árboles de directorios, que contienen los ficheros de cada *training* con las palabras claves.

### 4.4.2. GESTIÓN DE CUENTAS

Como ya se ha mencionado previamente, el acceso al repositorio se podrá llevar a cabo mediante interfaz web o mediante API REST. Pero, precisamente para ser capaz de recopilar información de este, se ha decidido que hay que disponer de cuenta con la que autenticarse. Se ha asumido que la solución disponga de dos tipos de cuenta. A continuación, se procederá a explicar las características de cada una de ellas y el motivo de esta decisión.

#### 4.4.2.1. CUENTA DE GOOGLE O GITHUB

Delegar la gestión de la cuenta a plataformas como Google o GitHub, proporciona una clara ventaja respecto a la administración, puesto que de esta manera se reducen los registros en las bases de datos de usuarios. Por otro lado, se ahorra espacio de almacenamiento en la futura base de datos que gestione este tipo de información. Si a esto se le añade, lo que se mencionaba anteriormente acerca de la facilidad de acceso, el hecho de poder registrarse con un simple *click* y no mediante largos formularios que requieren una gran cantidad de datos, es un punto bastante a favor en cuanto a la accesibilidad.

Las plataformas que se han decidido enlazar para permitir la autenticación son Google y GitHub. La primera de ellas debido a que hoy en día es raro encontrar a un usuario que no disponga de cuenta en esta institución. Tan solo con el hecho de ser usuario de Android, te obliga a tener cuenta en esta organización. Por otro lado, debido a la clase de público a la que está orientada la web y el actual crecimiento de GitHub, además de su API para autenticarse, hace que sea una opción bastante adecuada para implementarla como inicio de sesión.

En cuanto al nivel de acceso que tendrán los usuarios con este tipo de perfil, **se limitará únicamente a la interfaz web**, no pudiendo acceder de esta forma a las funcionalidades que otorgue la API REST. Esto es debido a factores de seguridad que se explicarán en detalle más adelante, pero a modo de resumen, puede establecerse que el sistema, al no gestionar los detalles de las credenciales de esas cuentas, no puede implementar mecanismos de autenticación en la API lo suficientemente seguros como para permitir su uso por este tipo de cuentas.

#### 4.4.2.2. CUENTA DEL SISTEMA

Este tipo de perfiles irán destinados a usuarios que pretendan realizar alguna clase de desarrollo a partir del repositorio de este sistema. Con ella, aparte de poder acceder libremente a la interfaz web, tendrán la posibilidad de autenticarse en la API REST, y por tanto, emplearla para sus propias implementaciones. En este caso, al disponer de sus credenciales, el sistema si puede garantizar un buen uso del mismo.

El procedimiento para llevar a cabo este registro es un poco tedioso para el usuario, sin embargo, elimina la opción de tener que desarrollar código que dé de alta al usuario en el sistema. El propio Django trae consigo un módulo de autenticación que, junto a la

interfaz de administrador, permite dar de alta a un usuario sin necesidad de un formulario.

Si un usuario desea hacer uso de la API, deberá mandar un email al administrador del sitio web indicando su propósito por el cual desea la cuenta. Éste, una vez lo reciba llevará a cabo el proceso de darle de alta a través de la interfaz web de administrador. Para ello, empleará el email que le indique el usuario y le añadirá una contraseña segura, que más tarde le reenviará al usuario. Una vez, el usuario haya recibido la contraseña el proceso habrá concluido y podrá utilizar satisfactoriamente tanto la interfaz web como la API.

### 4.4.3. INTERFAZ WEB.

En este módulo de la solución, se han centrado la mayoría de esfuerzos en cuanto a granularidad, seguridad y funcionalidad. Debido a que la sencillez apremia, se ha tratado de seguir un diseño que permita un uso fácil de la aplicación. Además, se han añadido vistas y herramientas dentro de las mismas, que permitan al usuario conocer el correcto funcionamiento del sistema.

A lo largo, de este punto, se expondrán todas aquellas subpáginas que se han incluido, con todas las decisiones que se hayan tenido que tomar en los diversos aspectos requeridos, con el fin de lograr los requisitos y objetivos expuestos.

#### 4.4.3.1. VISTA DE LA INTERFAZ WEB

Para el diseño de las plantillas HTML, se ha seguido un criterio muy básico para todas ellas. El texto estará contenido en un * * con fondo blanco semitransparente. Las letras serán de color negro, y se añadirá la tipografía que se determine más apropiada. En el caso de que en ellas se tengan que realizar alguna determinada acción, como por ejemplo la búsqueda, se incluirá en un * * con el mismo diseño, pero centrado en el centro de la página. Por último, en el *background*, se ha decidido mantener la misma imagen para todas las páginas.

En cuanto al dinamismo que se le ha añadido a la interfaz web, es necesario destacar que se ha desarrollado cierta lógica en JavaScript, concretamente en la vista de búsqueda avanzada, para permitir añadir dinámicamente más condiciones de búsqueda. Esto ha sido logrado mediante la implementación de 4 funciones, en las que se ha tenido que guardar un especial cuidado a la hora de generar los nuevos elementos, tomando vital importancia la nomenclatura empleada para generar los campos enviados al

servidor. Puesto que, en función de la misma, se realizará la lógica en el *back-end*, tal y como se verá en el punto 4.4.3.4.

#### 4.4.3.2. *LOGIN*

La autenticación de usuarios es uno de los aspectos clave en cuanto seguridad de cualquier aplicación. Mediante la implementación de este mecanismo se consiguen frenar usos indebidos de la aplicación. Es en este punto donde, se empiezan a utilizar todas las ventajas que proporciona Django.

Este *framework*, como ya se ha visto, posee la particularidad de que permite la instalación e integración de aplicaciones dentro del proyecto que se está implementando. Durante el desarrollo, se hará uso de varias de ellas. Sin embargo, en el caso de la autenticación, se empleará una sola: **"*social_django*"** [25]

Se ha decidido utilizar esta ya que, si bien no es la única que proporciona mecanismos de autenticación, es la que proporciona las funcionalidades más adecuadas al propósito de la solución. Esta aplicación, permite delegar el grueso de la gestión de cuentas a plataformas como Google, GitHub o diferentes redes sociales, para de esta forma, cumplir con el diseño establecido en la administración de cuentas.

El diseño de la vista se compondrá únicamente de dos botones que permitirán la creación de la cuenta y/o acceso de la misma al sistema. Además, dispondrá también de un formulario de inicio de sesión clásico en el que el usuario que posea cuenta del sistema podrá aplicar sus credenciales.

La implementación de la plantilla se ha llevado a cabo empleando código HTML junto a CSS para darle forma. En cuanto al desarrollo de código necesario para el proceso de autenticación, se ha hecho uso de la aplicación anteriormente mencionada y del propio módulo de autenticación que proporciona el *framework*.

Una vez generada la plantilla, lo único que ha sido necesario añadir ha sido la *url* relativa por la que se deseaba que se accediese a la aplicación web: *login/* y referenciarla a los recursos que pone Django a disposición para este fin.

```
url(r'^login/$', auth_views.LoginView.as_view(template_name = 'socialauth/login.html'), name = 'login'),
```

*Ilustración 20 - Código que indica la url por la que acceder al login y su función*

Siendo accesible ya la *url* del inicio de sesión, donde los usuarios podrán acceder al sistema por medio de la autenticación. Es precisamente este mecanismo, el que se

procede a implementar. Para ello, hay que hacer una serie de configuraciones en el fichero settings.py. Primeramente, añadir la aplicación *"social_django"* a la lista de aplicaciones instaladas, tal y como ha de hacerse con todas aquellas aplicaciones que se quieran emplear. Más tarde, es necesario añadir las siguientes variables:

```
AUTHENTICATION_BACKENDS = (
 'social_core.backends.open_id.OpenIdAuth',  # for Google authentication
 'social_core.backends.google.GoogleOpenId',  # for Google authentication
 'social_core.backends.google.GoogleOAuth2',  # for Google authentication
 'social_core.backends.github.GithubOAuth2',  # for Github authentication
 'django.contrib.auth.backends.ModelBackend',
)

SOCIAL_AUTH_GOOGLE_OAUTH2_KEY = '537500105189-6e22enhge9t3raa478vr8li0ke1bauem.apps.googleusercontent.com'
SOCIAL_AUTH_GOOGLE_OAUTH2_SECRET = 'pEJU5UNTDewSfjjZ2aU_OyR8'

SOCIAL_AUTH_GITHUB_KEY = '9f0d547371b304e1786e'
SOCIAL_AUTH_GITHUB_SECRET = '84ae8e1860eb1e67c46cb5c61d6f9b8b8896f10b'

LOGIN_REDIRECT_URL = 'search'
LOGOUT_REDIRECT_URL = 'logout'
```

*Ilustración 21 - Variables necesarias para la configuración de social_auth*

- **AUTHENTICATION_BACKENDS:** necesaria para comprobar la autenticación en las distintas plataformas.
- **SOCIAL_AUTH_GOOGLE/GITHUB_KEY:** la clave generada cuando se da de alta el servicio en la API de autenticación de las distintas plataformas.
- **SOCIAL_AUTH_GOOGLE/GITHUB_SECRET:** el secreto proporcionado al dar de alta la aplicación en la API de autenticación de las distintas plataformas.
- **LOGIN_REDIRECT_URL:** indica la *url* relativa a la que redirigir si la autenticación es exitosa. En este caso, se ha decidido que se redirija a la vista de búsqueda básica.
- **LOGOUT_REDIRECT_URL:** indica la *url* relativa a la que redirigir cuando se produzca el cierre de sesión.

Estas variables son necesarias para el correcto funcionamiento en la autenticación, sin embargo, el último paso consiste en añadir la *urls* de autenticación de las organizaciones externas, es decir, aquellas a las que habrá que redirigir al usuario cuando realicen el acceso mediante Google o GitHub. Para ello, hay que añadir la siguiente línea de código al fichero urls.py de la aplicación "*socialauth*".

```
url(r'^auth/', include('social_django.urls', namespace='social')),
```

*Ilustración 22 - Código que permite la redirección a la autenticación social*

De esta manera, se hace saber al intérprete de Django, que la autenticación va a correr a cargo de otros servicios. El desarrollo de esta funcionalidad se ha basado en recursos

que se han encontrado online. Entre los que destacan la propia documentación de Django [26], así como tutoriales [27].

Con todo esto, la funcionalidad de autenticación, creación de cuenta y acceso quedaría cubierta. Una vez realizada, el siguiente paso es diseñar e implementar la búsqueda básica.

#### 4.4.3.3. BÚSQUEDA BÁSICA

La primera página que verá el usuario tras iniciar sesión será la de búsqueda básica. En ella, será capaz de introducir texto, que será enviado al *back-end* para comprobar si existen coincidencias con algún campo de los metadatos o con alguna de las palabras. En el caso de que así sea, será redirigido a otra *url* donde se mostrarán por metadato o por término, los ejercicios que cumplan con la búsqueda. Aquellos que coincidan por término, se mostrará, aparte del título y la descripción, el número de veces que aparece el texto introducido, mientras que, los que sean encontrados por algún metadato, mostrarán únicamente título y descripción. Una vez diseñado el funcionamiento es hora de diseñar la implementación del mismo.

Por un lado, se dispondrá de dos plantillas HTML en las que se mostrará los datos obtenidos de la búsqueda. Por otro lado, se realizarán dos vistas en el fichero views.py del directorio *socialauth*. En estas vistas, se implementará la lógica correspondiente a la búsqueda y a la visualización de los ejercicios encontrados.

```
266  def search(request):
267      template = loader.get_template('search.html')
268      context = {"user":request.user}
269      return HttpResponse(template.render(context,request))
270
271  def found(request):
272      with open(os.path.join(BASE_DIR, '../data.json'),'r') as datafile:
273          data = json.load(datafile)
274      result = []
275      resultfiles = []
276      template = loader.get_template('found.html')
277      text = request.POST.get('text').lower()
278      trainings = basic_training_search(text,data)
279      result.append((text.upper(),trainings))
280      filestr = search_word_in_files(text,data)
281      resultfiles.append((text.upper(),filestr))
282      context = {'data':result, 'files':resultfiles}
283      return HttpResponse(template.render(context, request))
```

*Ilustración 23 - Código de las vistas de búsqueda básica y de muestra de resultados*

En el primer método (líneas 266-269 – Ilustración 23) únicamente se define la plantilla que se va a cargar en la *url* relativa a esta vista y se le añade al contexto el nombre de usuario para mostrarlo en la página.

En el segundo (líneas 271-283 – Ilustración 23) se cargan los datos del repositorio, se define la plantilla HTML que se va a cargar en la *url* que muestra los resultados de la búsqueda *(/found/*) y recoge los parámetros de la petición HTTP POST que se realiza al enviar el texto en la vista de búsqueda básica. Es en la línea 278 donde se produce la búsqueda básica con la llamada al método ***"basic_training_search(text,data)"***. El cual, recibe el campo de texto a buscar y los datos cargados del repositorio.

```
53  def basic_training_search(value,data):
54      result = []
55      trainingnames = []
56      text = []
57      for training in data['resources']:
58          #Preguntar primero si el training tiene la clave.
59          keys = list(training.keys())
60          for key in keys:
61          #Separar palabras de búsqueda
62              if type(training[key]) == str:
63                  if value.lower() in training[key].lower() and training not in result:
64                      result.append(training)
65                  else:
66                      continue
67
68              elif type(training[key]) == int:
69                  if value == str(training[key]):
70                      result.append(training)
71
72              elif type(training[key]) == list:
73                  for el in training[key]:
74                      if el.lower() == value and training not in result:
75                          result.append(training)
76                      else:
77                          continue
78      return result
```

*Ilustración 24 - Código correspondiente a la función basic_training_search*

En este método podemos observar cómo se itera por todos y cada uno de los materiales del repositorio. En cada uno de ellos se comprueba los metadatos que posee y se itera por cada uno de ellos. Se comprueba su tipo y en función del que sea se realiza la comparación con el valor aportado en el campo de texto. Si la comparación es exitosa, se almacena en una lista que al final de todas las iteraciones es devuelta, conteniendo todos aquellos ejercicios que hayan coincidido con la búsqueda.

```
162  def search_word_in_files(word,data):
163      trainingnames = []
164      result = []
165      PATH_DIR = os.path.join(BASE_DIR,'../')
166      directories = os.listdir(PATH_DIR)
167      for dirs in directories:
168          if "Files" in dirs:
169              dirtrainings = os.listdir(os.path.join(BASE_DIR,"../",dirs))
170              for trfolder in dirtrainings:
171                  file = os.path.join(BASE_DIR,"../",dirs,trfolder,'mainwordfile.txt')
172                  if os.path.exists(file):
173                      with open(file,'r') as wordsfile:
174                          repeated = 0
175                          for l in wordsfile:
176                              match = re.findall(word,l.lower())
177                              repeated = len(match)
178                          if repeated > 0:
179                              dirsname = file.split("/")
180                              print(dirsname)
181                              trainingnames.append((dirsname[-2],repeated))
182
183      for training in data['resources']:
184          for tr in trainingnames:
185              if tr[0] == training['title']:
186                  result.append([training,tr[1]])
187                  result.sort(key=lambda numword: numword[1], reverse = True)
188
189      return result
```

*Ilustración 25 - Código que busca un término dado en los ficheros mainwordfile.txt*

En este fragmento de código se refleja la codificación realizada para el método llamado en la línea 280 de la Ilustración 23. En él, se accede a la raíz del sistema de ficheros de la aplicación. Ahí se listan todos los directorios y se accede a aquellos que contengan la palabra "*Files*", de tal forma que, si se añaden nuevas fuentes de datos, mientras se mantenga la misma nomenclatura, será escalable. Una vez ahí, entra en cada una de las carpetas correspondientes a las fuentes de datos e itera por cada uno de los directorios con nombre de ejercicios. Una vez dentro, accede al fichero generado por el script que analiza los términos y busca coincidencias dentro de él. En caso de que haya una coincidencia añade el nombre del directorio a una lista y el número de coincidencias. Para más tarde, en las líneas 183-187 (Ilustración 25), iterar por el repositorio para recoger todos los datos correspondientes al training con ese nombre y ordenarlos por el número de apariciones del texto en el fichero de términos. Por último, devolverá la lista con todos los trainings que cumplan el criterio, ordenados.

Para terminar el método "***found***", se añade al contexto la lista de los ejercicios que satisfacen la búsqueda en cuanto a metadatos o en cuanto a términos y devuelve la respuesta HTTP en la que la variable contexto se empleará para la construcción dinámica de la página HTML.

Por último, explicadas ambas vistas, es hora de detallar las configuraciones necesarias para hacerlas accesibles. El fichero de urls.py de *socialauth* (de ahora en adelante, a lo largo del punto 4.4.3 se asumirá que todos los ficheros mencionados hacen referencia a

la aplicación *socialauth*, a no ser que se indique lo contrario) ha de ser modificado añadiéndole las siguientes líneas para permitir su localización por mediante url.

```
path('', search, name = 'search'),
                    url(r'^found/', found, name='found'),
```

*Ilustración 26 - Asignación de las urls de búsqueda simple y mostrar ejercicios*

#### 4.4.3.4. BÚSQUEDA AVANZADA.

La búsqueda avanzada es, sin duda, uno de los aspectos más complejos de este proyecto. Por un lado, ha de hacerse para que pueda ser escalable a medida que se añaden nuevas fuentes de datos. Por otro lado, ha de ser lo suficientemente intuitiva para que el usuario sea capaz de manejarla correctamente. Esta funcionalidad, estará basada en dos vistas de Django y dos plantillas HTML con, tal y como se ha comentado antes, bastante presencia de código JavaScript.

La primera de estas vistas, será la correspondiente a mostrar la búsqueda avanzada. Para ello, será necesario hacer una llamada a la plantilla que vaya a cargar, y pasarle en el contexto aquellos filtros que deseen visualizarse mediante *checkbox* o desplegable. La condición para que un metadato se muestre de una forma u otra es que, si la variedad de su contenido es inferior a cinco valores distintos, se mostrarán en un *checkbox*, que al seleccionarlo, incluya un desplegable con dichos valores. Esta funcionalidad se hará por medio del método ***def get_filters()*** de la línea 279 (Ilustración 27).

```
277  def advancedsearch(request):
278      template = loader.get_template('advancedsearch.html')
279      context = get_filters()
280      return HttpResponse(template.render(context,request))
```

*Ilustración 27 - Código de la vista de búsqueda avanzada*

La *template* referenciada en la imagen superior, ***advancedsearch.html***, será la destinada al buscador. Ésta, permitirá buscar si el valor introducido está contenido o es exactamente ese valor. Además, podrás añadir más entradas con las que realizar búsquedas más complejas con operadores lógicos *AND* y *OR*. En la siguiente ilustración, se podrá observar, primeramente, el estado del buscador en su momento inicial y más tarde al activar los *checkboxes* y añadir un criterio más.

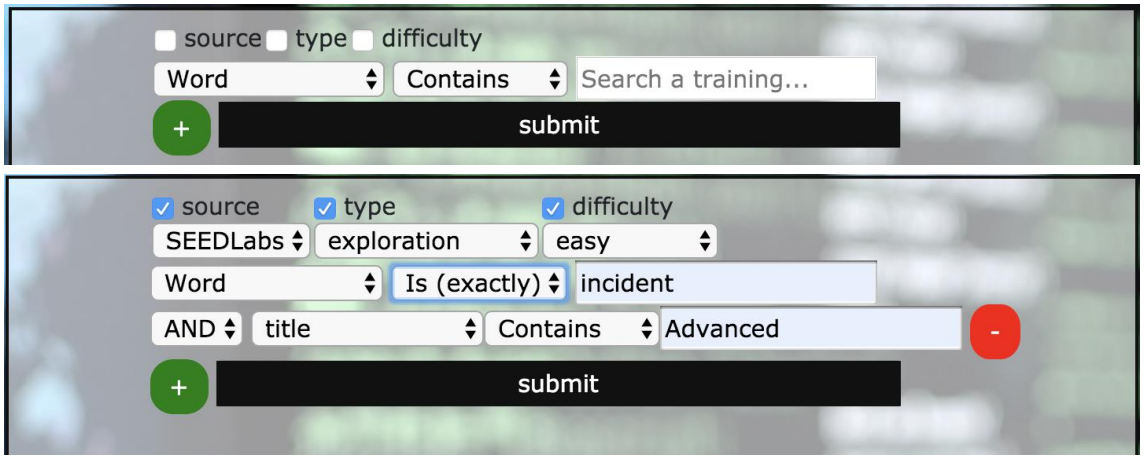
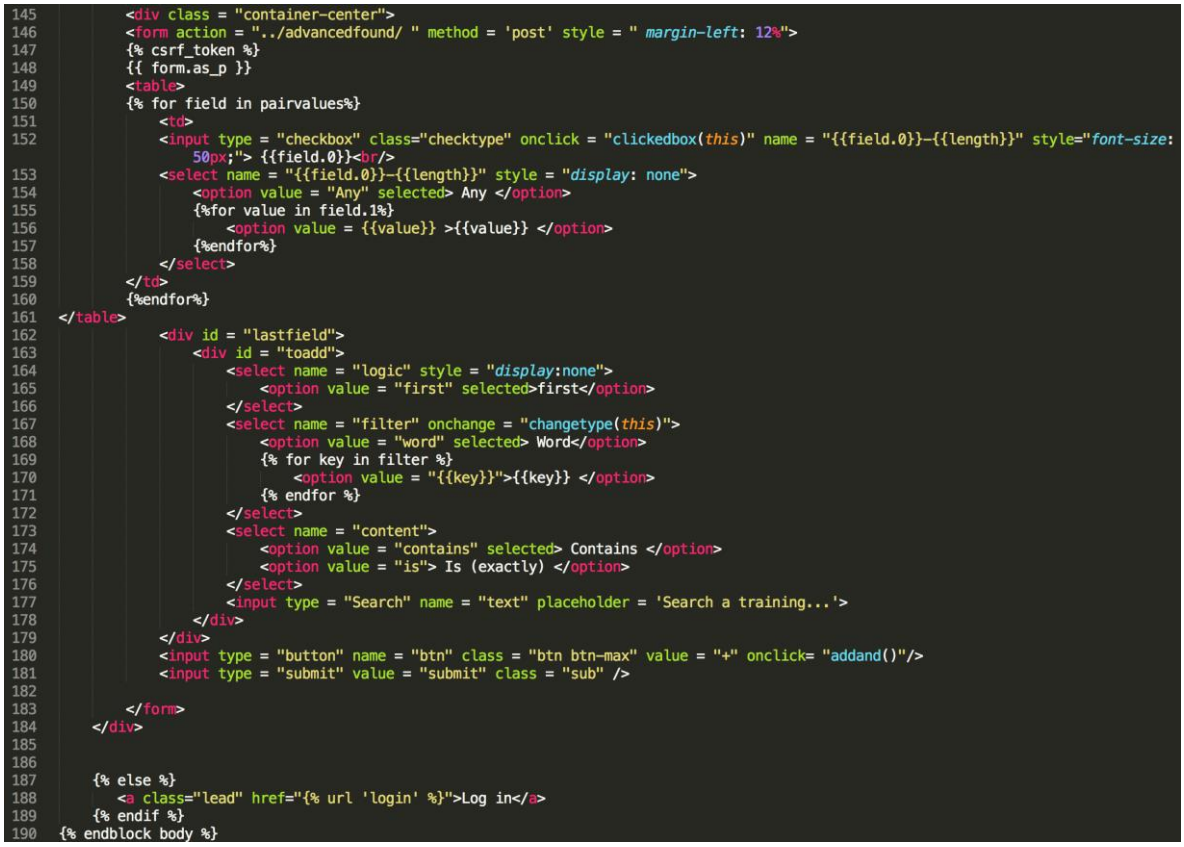



*Ilustración 28 - Ejemplo de cómo se visualiza la búsqueda avanzada*

La funcionalidad propia de los *checkboxes* y el añadir más criterios de búsqueda es lo que ha sido desarrollado en JavaScript. Se ha hecho de tal forma que sea escalable y sin límite por lo que se podrán añadir tantos como se quiera.

En la imagen inferior, la Ilustración 29, puede apreciarse como se ha implementado el código HTML. Es necesario adjuntarla a este documento, debido a que se han tenido que tomar bastantes decisiones de diseño en ella. Por un lado, la nomenclatura empleada en los campos que van a ser enviados al servidor con la información de la búsqueda avanzada. Tal y como se ha visto en las imágenes anteriores, los campos que resultan de utilidad mandar al servidor son, el metadato por el que se va a hacer la búsqueda, si se pretende que esté contenido o sea exacto y el texto a buscar, y en el caso de que se añada un criterio, la lógica que se desee seguir, es decir si es *AND* u *OR*. Además, hay que tener en cuenta si el *checkbox* ha sido marcado o no, de tal forma que si ha sido marcado enviará el dato seleccionado y si no, la cadena "*Any*" que indicará que no se ha seleccionado ningún valor para el mismo.



```
        <div class = "container-center">
        <form action = "../advancedfound/ " method = 'post' style = " margin-left: 12%">
        {% csrf_token %}
        {{ form.as_p }}
        <table>
        {% for field in pairvalues%}
            <td>
            <input type = "checkbox" class="checktype" onclick = "clickedbox(this)" name = "{{field.0}}-{{length}}" style="font-size:
                50px;"> {{field.0}}<br/>
            <select name = "{{field.0}}-{{length}}" style = "display: none">
                <option value = "Any" selected> Any </option>
                {%for value in field.1%}
                    <option value = {{value}} >{{value}} </option>
                {%endfor%}
            </select>
        </td>
        {%endfor%}
</table>
            <div id = "lastfield">
                <div id = "toadd">
                    <select name = "logic" style = "display:none">
                        <option value = "first" selected>first</option>
                    </select>
                    <select name = "filter" onchange = "changetype(this)">
                        <option value = "word" selected> Word</option>
                        {% for key in filter %}
                            <option value = "{{key}}">{{key}} </option>
                        {% endfor %}
                    </select>
                    <select name = "content">
                        <option value = "contains" selected> Contains </option>
                        <option value = "is"> Is (exactly) </option>
                    </select>
                    <input type = "Search" name = "text" placeholder = 'Search a training...'>
                </div>
            </div>
            <input type = "button" name = "btn" class = "btn btn-max" value = "+" onclick= "addand()"/>
            <input type = "submit" value = "submit" class = "sub" />

        </form>
    </div>


    {% else %}
        <a class="lead" href="{% url 'login' %}">Log in</a>
    {% endif %}
{% endblock body %}
```

*Ilustración 29 - Código HTML de la vista de la búsqueda avanzada*

La nomenclatura que se ha decidido emplear es la siguiente:

- ***logic*:** indicará *first* cuando se trate del primer criterio, y *and* u *or* cuando se añadan más criterios y en función de la lógica que apliquen.
- ***filter*:** indicará el metadato o término por el que se hará la búsqueda. Estarán añadidos al desplegable dinámicamente empleando los mecanismos que otorga Django para ello, con el fin de que sea escalable.
- ***content*:** indicará si se pretende que la búsqueda se haga por valor contenido o exacto.
- ***text*:** indicará el texto introducido por el usuario, el cual se pretende encontrar en el campo especificado en *filter*.

Para desarrollar el dinamismo y que a su vez sea efectivo en el servidor, a la nomenclatura anterior se ha decidido añadirle un contador, de tal forma que el código JavaScript lleve la cuenta de los criterios introducidos y le añada a la nomenclatura el valor del contador en cada momento que se pinche en el botón (+).

Una vez visto la lógica y el diseño realizado en el *front-end* es hora de trasladar las explicaciones a la capa del servidor. La vista que se encarga de realizar la búsqueda avanzada, está denominada como ***def advancedfound(request)*** en el fichero "*views.py*"

de la aplicación *socialauth*. Al tratarse de un método muy extenso, se va a proceder a dividirlo explicando aquellos aspectos que resulten más relevantes a nivel de diseño e implementación.

El método comienza cargando el repositorio en un diccionario de Python para poder realizar operaciones sobre él y definiendo las variables que se van a utilizar, entre las que cabe destacar, la lista de los nombres de los parámetros recogidos en la petición que recibe. A continuación, ejecuta lo descrito en la siguiente imagen:

```
300     for param in parameters:
301         if "-" in param:
302             fixedsearch.append(param)
303     print(fixedsearch)
304     for p in fixedsearch:
305         print("{0} eliminado".format(p))
306         parameters.remove(p)
307     while(len(fixedsearch) != 0):
308         key = fixedsearch.pop()
309         value = request.POST.get(key)
310         key = key.split('-')[0]
311         print("La clave es: {0} y el valor: {1}".format(key,value))
312         if(value != "Any"):
313             dataset[key] = []
314             for training in data['resources']:
315                 if key in list(training.keys()):
316                     if training[key] == value:
317                         if training not in dataset[key]:
318                             dataset[key].append(training)
319             checkboxclicked+= 1
320 
321     if checkboxclicked != 0:
322         andtrainings = get_checkbox_repeated(dataset)
323         if len(andtrainings[0]) == 0:
324             template = loader.get_template("emptylist.html")
325             return HttpResponse(template.render(context,request))
326     else:
327         andtrainings = (data['resources'],0)
```

*Ilustración 30 - Primer fragmento de código de la función advancedfound*

Itera por toda la lista de parámetros en busca de aquellos que sean referentes a los *checkboxes*. Tal y como se puede observar en la figura del código HTML (Ilustración 29), todos los *checkbox* dispondrán en su nomenclatura un "-" que se empleará para detectar precisamente que se tratan de *checkboxes*. Una vez establecidos qué parámetros son los correspondientes a éstos, se añadirán a una lista auxiliar (líneas 300-306 – Ilustración 30) que más tarde, se iterará (líneas 307-319 – Ilustración 30) realizando la búsqueda de los valores seleccionados en los *checkboxes* y actualizará un contador en caso de que se haya elegido un valor distinto a *"Any"*. Si efectivamente, se ha elegido algún valor del *checkbox*, se llamará al método ***"get_checkbox_repeated(dataset)"*** en una variable que, posteriormente, se encargará de limitar el diccionario anteriormente cargado a los

criterios de los *checkboxes*. Si el resultado de la llamada a ese método es una lista sin elementos, se redirigirá a una vista en la que se notifica al usuario que no hay coincidencias en su búsqueda. En caso de que ningún *checkbox* haya sido activado, se seguirá empleando el diccionario originalmente cargado desde el fichero JSON.

```
331             data['resources'] = andtrainings[0]
332             parametersaux = parameters
333             parametersaux.remove('csrfmiddlewaretoken')
334             parametersaux.sort()
335             print(parametersaux)
336             groupsearch = get_post_parameters(parametersaux)
```

*Ilustración 31- Segundo fragmento de código de la función advancedfound*

En el código superior, se puede apreciar como efectivamente se limita el diccionario anteriormente cargado del fichero JSON, a los *trainings* obtenidos mediante los *checkboxes*. Además, se genera una lista de parámetros auxiliar sobre la que hacer operaciones. Finalmente, se llama al método "***def get_post_parameters(parametersaux)"*** que se encuentra codificado de la siguiente manera:

```
247 def get_post_parameters(parameters):
248     groupsearch = []
249     x = len(parameters)
250     while (x != 0):
251         parameterlist = [parameters.pop(),parameters.pop(),parameters.pop(),parameters.pop()]
252         print(parameterlist)
253         groupsearch.append(parameterlist)
254         x = len(parameters)
255     groupsearch.sort()
256     return groupsearch
```

*Ilustración 32 - Código correspondiente a la función get_post_parameters*

De lo que se encarga dicha función es de, habiendo recibido una lista de parámetros, iterar sobre ella hasta que quede vacía agrupando en una lista un conjunto de listas que posea en cada elemento el nombre de los parámetros vistos anteriormente respecto a la lógica, el metadato, el tipo de búsqueda (*contains* o *is*) y el texto introducido. De esta manera, cuando el usuario introduzca más de un criterio en la búsqueda, retornará una lista con el mismo número de criterios, en la que, a su vez, cada elemento constituirá en sí mismo una lista con los nombres de los cuatro parámetros asociados al criterio. Este método es necesario debido a la funcionalidad que se ha visto, implementada mediante JavaScript, en la que a la nomenclatura establecida se le añadía el valor de un contador para diferenciar los distintos criterios de búsqueda.

A continuación, en la imagen inferior, con la lista obtenida se harán una serie de peticiones, para con el nombre del parámetro, obtener su valor correspondiente. De tal forma que se generará otra lista con las mismas dimensiones que en lugar de contener

el nombre de los parámetros, contendrá su valor (líneas 338-344 – Ilustración 33). Más tarde, en las líneas 353-364, se iterará por esa lista que contiene el conjunto de listas referentes a cada criterio y se llevará a cabo la búsqueda, guardando en cada iteración del bucle, en dos listas los resultados obtenidos para cada criterio.

```
338         for parameter in groupsearch:
339             search = ["","","",""]
340             search[0] = request.POST.get(parameter[3])
341             search[3] = request.POST.get(parameter[0]).lower()
342             search[2] = request.POST.get(parameter[1])
343             search[1] = request.POST.get(parameter[2])
344             finalsearch.append(search);
345 
346         finalsearch.reverse()
347         x = len(finalsearch)
348         print(finalsearch)
349         numlogics = x
350         print("LA LONGITUD DE FINALSEARCH ES: {0}".format(x))
351         advancedlogic = []
352         query = "The search: "
353         while(x != 0):
354             key = finalsearch[x-1][1]
355             value = finalsearch[x-1][3].lower()
356             contains = finalsearch[x-1][0]
357             logic = finalsearch[x-1][2]
358             result = []
359             result = basic_key_search(key,value,contains,logic,data)
360 
361             advancedsearchtrs.append(result[1])
362             advancedlogic.append(result[0])
363             x = x-1
364             query += logic + " " + key + " " + contains + " " + value + " "
365 
```

*Ilustración 33 - Tercer fragmento de código de la función advancedfound*

En la imagen inferior (Ilustración 34), se puede apreciar cómo se realiza la búsqueda del método "***def basic_key_search(key,value,typesearch,logic,data)***", el cual comprueba el tipo de búsqueda que sigue el criterio (se describirá para el caso en el que se quiera buscar que el texto este contenido, en el caso de que se quiera buscar su valor exacto, sería la misma lógica pero aplicando la igualdad en lugar de la palabra reservada "*in*"). Más tarde, se comprueba si se pretende encontrar por término o por metadato. En el caso de que se pretenda buscar por metadato, buscará entre el conjunto de datos introducido por parámetro una coincidencia de clave: valor. Si la encuentra, añadirá el ejercicio a una lista que retornará como resultado, en la que la primera posición indicará la lógica que se aplicará. Si lo que busca es un término, entonces hará uso del método visto anteriormente en la imagen, en el que busca la coincidencia en los ficheros de términos contenidos en los árboles de directorios.

```
80  def basic_key_search(key,value,typesearch,logic,data):
81      advancedsearchtrs = []
82      result = []
83      if typesearch == "contains":
84          if key != "word":
85              for training in data['resources']:
86                  if training not in result:
87                      keys = list(training.keys())
88                      if key in keys:
89                          if type(training[key]) == str:
90                              if value.lower() in training[key].lower():
91                                  result.append(training)
92                              else:
93                                  continue
94                          elif type(training[key]) == int:
95                              if value in str(training[key]):
96                                  result.append(training)
97
98                          elif type(training[key]) == list:
99                              for el in training[key]:
100                                 if el.lower() == value:
101                                     result.append(training)
102                                 else:
103                                     continue
104             advancedsearchtrs.append(result)
105             advancedsearchtrs.insert(0,logic)
106             return advancedsearchtrs
107         else:
108             result = search_word_in_files(value,data)
109             resultfiles = []
110             for tr in result:
111                 resultfiles.append(tr[0])
112                 print("File {0}".format(tr[0]['title']))
113                 advancedsearchtrs.append(resultfiles)
114             advancedsearchtrs.insert(0,logic)
115             return advancedsearchtrs
116     elif typesearch == "is":
```

*Ilustración 34 - Código correspondiente a la función basic_key_search*

Volviendo al código anterior (Ilustración 33), una vez se ha realizado la búsqueda se procede a armar el *string* que se empleará para indicar al usuario la búsqueda que ha introducido. Además, añadirá a una lista los trainings encontrados y a otra la lógica que ha de aplicarles. Para que, cuando finalice el bucle, sea el momento de aplicarla en la búsqueda.

La forma de ejecutar la búsqueda avanzada consistirá en agrupar aquellos criterios cuya lógica sea *AND.* En caso de que exista un criterio con lógica OR, se tomará como el limitador para constituir otra agrupación de criterios *AND*. En la imagen inferior se podrá encontrar un esquema del funcionamiento.

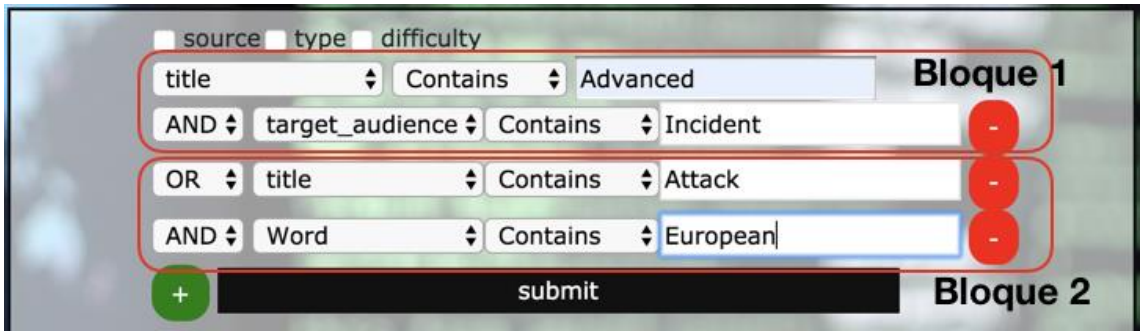



*Ilustración 35 - Ejemplo de búsqueda avanzada con varios criterios*

Puede por tanto entenderse, que la búsqueda, primeramente, resolverá cada uno de los criterios de manera individual, para después agruparlos como se ha descrito y tal y como puede apreciarse en los bloques de la figura.

De esta manera, atendiendo al ejemplo de la Ilustración 35, puede verse como se realizaría la búsqueda de todos aquellos ejercicios cuyo título contuviese la palabra *Advanced* y el público objetivo contuviese la palabra *Incident*, constituyendo así el BLOQUE 1. Al cual se le realizaría una operación de *OR*, con los resultados de los materiales cuyo título contenga *Attack* y en sus documentos esté contenida la palabra European, BLOQUE 2.

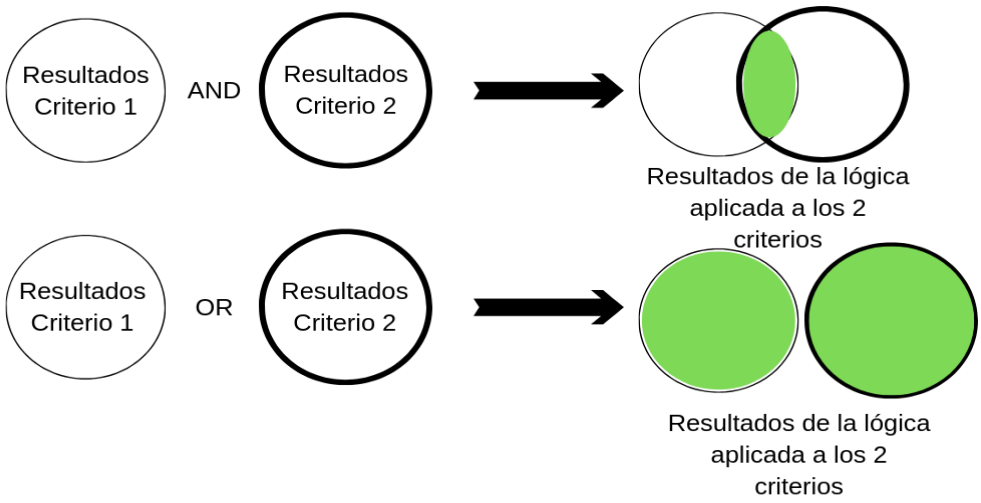



*Ilustración 36 - Esquema de las lógicas OR y AND*

```
368         element = 0
369         while element < len(advancedlogic):
370             if advancedlogic[element] == "or":
371                 indexsplit.append(element)
372             element+=1
373 
374         for index in range(len(indexsplit)):
375             if len(resultr) == 0:
376                 resultr.append(advancedsearchtrs[:indexsplit[index]])
377             if index+1 in range(len(indexsplit)):
378                 resultr.append(advancedsearchtrs[indexsplit[index]:indexsplit[index+1]])
379             else:
380                 resultr.append(advancedsearchtrs[indexsplit[index]:])
381 
382         andtrainings = []
383         finalresultand = []
384         finalresult = []
385         alltrainings = []
386 
387         if(len(resultr) > 0):
388             print("Longitud de resultr: {0}".format(len(resultr)))
389             for andtrs in resultr:
390                 andtrainings = []
391                 numands = len(andtrs)
392                 print("Numero de ANDS: {0}".format(numands))
393                 for andresult in andtrs:
394                     print("Longitud de andresult: {0}".format(len(andresult)))
395                     for tr in andresult:
396                         andtrainings.append(tr)
397                 alltrainings.append([andtrainings,numands])
398 
399             for res in alltrainings:
400                 for tr in res[0]:
401                     print()
402                     if res[0].count(tr) == res[1]:
403                         if tr not in finalresultand:
404                             finalresultand.append(tr)
405             print("La longitud de todos los trainigs: {0}".format(len(alltrainings)))
406 
407         else:
408             numands = len(advancedsearchtrs)
409             print("Numero de ANDS: {0}".format(numands))
410             for res in advancedsearchtrs:
411                 for tr in res:
412                     finalresult.append(tr)
413             for tr in finalresult:
414                 if finalresult.count(tr) == numands:
415                     if tr not in finalresultand:
416                         finalresultand.append(tr)
417 
418         template = loader.get_template('foundlogic.html')
419         context['data'] = finalresultand
420         context['text'] = query
421 
422         return HttpResponse(template.render(context, request))
```

*Ilustración 37 - Cuarto fragmento del código de la función advancedfound*

En la imagen superior, puede verse como se ha implementado la lógica de la búsqueda. En las líneas 369-372 – Ilustración 37, se procede a almacenar en una lista, el índice en el que se encuentran las lógicas *OR*, para de esta forma poder separar las agrupaciones de criterios en *ANDS.* Dicha separación en agrupaciones es justamente lo que se codifica en las líneas 374-380. En ellas, se rellena una lista compuesta a su vez de otras listas que incluyen los resultados de cada criterio en agrupaciones de *ANDS*. Finalmente, de la 387 a la 416 se comprueba que los *trainings* de cada agrupación estén presentes en los resultados de cada criterio que conforma la agrupación. De ser así, se añade a la lista final con el objetivo de mostrarlo en la *template* indicada en la línea 418.

Por último, una vez que ya se ha realizado la búsqueda, es necesario realizar la vista de los resultados de la misma. Para ello se ha empleado la *template* "**foundlogic.html**", que los representará en forma de tabla, indicando su título y descripción. En ella, pinchando en el título redirigirá a la vista que se verá en el punto 4.4.3.5, el detalle del ejercicio.

Antes de acabar, para hacer accesible ambas vistas es necesario adjudicarles una url relativa por las que encontrarlas, para ello, se han añadido las siguientes líneas al fichero ***urls.py*** siendo la primera de ellas la correspondiente a la búsqueda avanzada y la segunda la correspondiente a los resultados de la misma.

```
14      url(r'^advanced/$', advancedsearch, name = 'advanced'),
15      url(r'^advancedfound/$', advancedfound, name = 'advancedfound'),
```

*Ilustración 38 - Urls de la búsqueda avanzada y la visualización de los resultados*

#### 4.4.3.5. DETALLE DEL EJERCICIO

En esta vista se detallarán todos los datos comunes a cada uno de los *trainings* de cada fuente, tomando por tanto la siguiente estructura:

1. **Id - Título:** constituirá la cabecera de la página. Tal y como se podrá intuir, hará uso de los metadatos correspondientes al id y al título del training, así como al campo "*site*", que será incluido como hipervínculo para poder redirigir al usuario hacia la vista oficial del ejercicio, propia de la fuente de datos.
2. **Fuente:** debajo del título, referenciará el origen del *training.*
3. **Descripción:** detallará la descripción obtenida haciendo uso del metadato "description", común a todas las fuentes de datos.
4. **Duración:** indicará la duración estimada tomada del campo "*duration*".
5. **Documentos:** en este campo se listarán todos los ficheros necesarios para la resolución del ejercicio. Además, se añadirá como hipervínculo sus *urls* asociadas por lo que empleará los metadatos referentes a *"files"* y "*urls".*

Esto en cuanto a la plantilla, pero en cuanto a la vista propia del fichero ***views.py*** es necesario tener en cuenta ciertos factores. El más importante es decidir de qué forma se va a identificar el ejercicio a mostrar. Para ello, se ha decidido que, cuando se muestre el conjunto de trainings para una búsqueda dada, el título de cada uno de ellos, contenga un hipervínculo que introducirá en la *url* como parámetro el id que le referencia. De esta forma, cuando en el *back-end* llegue la petición HTTP(s) se podrá recoger el parámetro

y determinar cuál es el *training* del que se desea conocer más información. Todo esto se lleva a cabo de la siguiente manera.

```
426  def showtraining(request):
427      with open(os.path.join(BASE_DIR, '../data.json'),'r') as datafile:
428          data = json.load(datafile)
429      template = loader.get_template('showtraining.html')
430      trainingid = request.GET.get('id')
431      context = {}
432      trainingfiles = []
433      files = []
434      for tr in data['resources']:
435          if str(tr['id']) == trainingid:
436              training = tr
437      rango = range(len(training['files']))
438      linksname = []
439      for x in rango:
440          linksname.append([training['files'][x],training['urls'][x]])
441      context = {'training': training, 'linksname':linksname}
442      return HttpResponse(template.render(context,request))
```

*Ilustración 39 - Código correspondiente a la función showtraining*

Se cargan los datos del fichero JSON, más tarde, se consigue el parámetro id de la *url* al ser un método post. Finalmente, se hace la búsqueda por id en los datos leídos del fichero y aquel que coincida se pasa al contexto, junto con una lista que guarda la relación entre los ficheros y las *urls* donde encontrarlos. Todo ello da como resultado la vista de la siguiente imagen.


*Ilustración 40 - Visualización del ejercicio en la interfaz web*

De nuevo y, por último, es necesario hacer accesible esta vista, para ellos se añade la *url* relativa al fichero de ***urls.py***:

```
url(r'^show/$', showtraining, name = 'showtraining'),
```

*Ilustración 41 - Url correspondiente a la visualización de ejercicios*

#### 4.4.3.6. ABOUT US

En esta sección, no requerirá apenas de desarrollo de código, puesto que su objetivo es ser meramente informativa. En ella, se incluirá información acerca del proyecto y de sus integrantes.

```
450   def aboutus(request):
451       return render(request,'aboutus.html')
```

*Ilustración 42 - Código correspondiente a la función aboutus*

Esas líneas, junto a la configuración de la *url* en el fichero correspondiente, es todo lo necesario para hacer accesible la vista y por tanto permitir que el usuario conozca más acerca de los detalles del proyecto.

```
url(r'^aboutus/$', aboutus, name='aboutus'),
```

*Ilustración 43 - Url que corresponde a la visualización de la ventana about us*

#### 4.4.3.7. USEFUL INFORMATION

Esta página, al igual que la anterior, no requerirá de un gran número de líneas de código. En ella, figurará toda la información relevante al funcionamiento del motor de búsqueda. Así como, un manual para desarrolladores, donde vendrá explicada el procedimiento que ha de seguirse para un correcto uso de la API REST. La vista correspondiente codificada es la siguiente:

```
444   def usefulinfo(request):
445       info = get_filters()
446       context = {'info':info}
447       template = loader.get_template("usefulinfo.html")
448       return HttpResponse(template.render(context,request))
```

*Ilustración 44 - Código correspondiente a la función usefulinfo*

Puede observarse como únicamente recoge información acerca de las claves que se pueden emplear para la búsqueda, mediante el método "***get_filters()***", con el fin de mostrárselo al usuario que tenga interés en desarrollar software haciendo uso de la API.

Añadiendo las siguientes líneas al fichero de **urls.py** se logra que sea alcanzable a través de la url: https://localhost:8000/usefulinfo.

```
url(r'^usefulinfo/$', usefulinfo, name='usefulinfo'),
```

*Ilustración 45 - Url de la visualización de la ventana de useful information*

#### 4.4.3.8. *LOG OUT*

Constituye el fin del uso que el usuario hace del sistema, mediante esta vista, será capaz de cerrar su sesión de manera segura. Para su implementación se ha empleado el método "***logout***" propio de Django y la *url* correspondiente a la función, quedando codificada de la siguiente forma.

```
453  def logout_view(request):
454      logout(request)
455      return render(request,'socialauth/logout.html')
```

*Ilustración 46 - Código correspondiente a la función logout_view.*

```
url(r'^logout/$', logout_view, name = 'logout'),
```

*Ilustración 47 - Url de la funcionalidad de cierre de sesión*

### 4.4.4. API REST

A continuación, se tratará el módulo de la solución destinada a desarrolladores. La API REST permitirá mediante peticiones HTTPS, autenticarse en el sistema y realizar una búsqueda simple.

Se ha diseñado de manera muy sencilla, puesto que supondrá el inicio de la funcionalidad general. Dentro del proyecto de Django, está contenida en el directorio "*APIfaceREST*". El cual contiene los ficheros views.py, urls.py, etc. necesarios para la configuración de la aplicación en el entorno del proyecto. Al tratarse de una API REST no requerirá de vistas, pero sí de una *url* mediante la cual pueda ser llamada. Para ello es necesario modificar el fichero ***urls.py*** con las siguientes líneas:

```
1  from django.conf.urls import url
2  from APIfaceREST import views
3  urlpatterns = [
4      url(r'^trainings/$', views.traininglist),
5  ]
```

*Ilustración 48 - Url para la funcionalidad de la API REST*

Una vez añadida la *url* por la que pueden ser solicitados los recursos, sólo queda añadirle la lógica para hacerlo posible. Los siguientes apartados tratarán aquellos detalles del diseño e implementación de los aspectos más relevantes de esta parte de la solución.

#### 4.4.4.1. AUTENTICACIÓN

La autenticación de la cuenta en la API es uno de los factores más críticos. Es necesario proteger el acceso a nuestro sistema, y al usuario, de forma que no se produzcan usos indebidos. Para la autenticación en este tipo de tecnologías que hacen uso de HTTP(S), se hace uso de la cabecera ***Authorization*** propia de las peticiones de dicho protocolo. Este campo acepta un amplio rango de valores, dentro de los cuales su funcionalidad va variando. Por mencionar algunos estudiados a lo largo del grado, se destacarán:

- ***Authorization: Basic*****:** Tal y como su propio nombre indica es el mecanismo más básico de todos para la autenticación. En él, las credenciales son proporcionadas codificadas en base64. La cadena ha de estar compuesta por el usuario del *login*, más el caracter ":" y por último la contraseña. Como se puede deducir, no es muy seguro al conocerse la codificación con la que está encriptado. Es por este motivo por lo que queda descartado.

- ***Authorization: Digest*****:** Es más seguro que el anterior, puesto que, a grandes rasgos, envía las credenciales cifradas según un código proporcionado por el servidor (*nonce*). Sin embargo, esta autenticación requiere bastantes esfuerzos a la hora de implementar la funcionalidad.

Vistos los conocimientos en cuanto autenticación adquiridos y sus limitaciones, se estudiaron otros mecanismos de autenticación. Durante este ejercicio, se pudo observar que la mejor opción en cuanto a seguridad y facilidad de implementación es emplear *JSON Web Token Authorization (JWT).* Está basado en el ***Authorization: Bearer***, el cual requiere un *token* para controlar la autorización. Dicho *token*, estará firmado por la clave del servidor, así que el cliente y el servidor son ambos capaces de verificar que el *token* es legítimo [28]. Una vez se disponga del *token* de acceso, todas las consultas que se le

hagan al servidor deberán incluir la cabecera ***Authorization: Bearer <tokenJWT>***. De esta forma, el servidor será consciente de que el cliente está autenticado y podrá servirle la información que requiera.

Django dispone de una aplicación que permite la autenticación mediante estos mecanismos. Se llama *"**rest_framework**"* y concretamente para la autorización mediante JWT, es necesario instalar una extensión: ***"djangorestframework_simplejwt"*** [29]. Para su configuración es necesario añadir lo siguiente al fichero *settings.py*

```
REST_FRAMEWORK = {
 'DEFAULT_AUTHENTICATION_CLASSES': ( 'rest_framework_simplejwt.authentication.JWTAuthentication',),
 'DEFAULT_PERMISSION_CLASSES': (
        'rest_framework.permissions.IsAuthenticated',
    )

}

SIMPLE_JWT = {
    'ACCESS_TOKEN_LIFETIME': timedelta(minutes=60),
    'REFRESH_TOKEN_LIFETIME': timedelta(days=1),
    'ROTATE_REFRESH_TOKENS': False,
    'BLACKLIST_AFTER_ROTATION': True,

    'ALGORITHM': 'HS256',
    'SIGNING_KEY': SECRET_KEY,
    'VERIFYING_KEY': None,

    'AUTH_HEADER_TYPES': ('Bearer',),
    'USER_ID_FIELD': 'id',
    'USER_ID_CLAIM': 'user_id',

    'AUTH_TOKEN_CLASSES': ('rest_framework_simplejwt.tokens.AccessToken',),
    'TOKEN_TYPE_CLAIM': 'token_type',

    'JTI_CLAIM': 'jti',

    'SLIDING_TOKEN_REFRESH_EXP_CLAIM': 'refresh_exp',
    'SLIDING_TOKEN_LIFETIME': timedelta(minutes=5),
    'SLIDING_TOKEN_REFRESH_LIFETIME': timedelta(days=1),
}
```

*Ilustración 49 - Variables de configuración para la autenticación JWT*

En líneas generales, lo que configura este fragmento de código es la autenticación mediante JWT, así como todas aquellas variables referentes al *"refresh"* de los *tokens*, el algoritmo, la clave secreta y demás constantes. Si se desease ampliar el alcance de la seguridad, se podría consultar el manual [29] donde se establecen más variables con estos fines.

Además, será necesario también modificar el archivo ***urls.py*** de "*cysectrainingface*", con el objetivo de habilitar las *urls* para solicitar el acceso. Dicho fichero, quedaría finalmente codificado de la siguiente manera:

```
1  from django.conf.urls import url, include
2  from django.contrib import admin
3  from rest_framework_simplejwt.views import (
4      TokenObtainPairView,
5      TokenRefreshView,
6  )
7
8  urlpatterns = [
9      url(r'^admin/', admin.site.urls),
10     url('', include('socialauth.urls')),
11     url(r'^',include('APIfaceREST.urls')),
12     url(r'^api/token/$', TokenObtainPairView.as_view(), name='token_obtain_pair'),
13     url(r'^api/token/refresh/$', TokenRefreshView.as_view(), name='token_refresh'),
14 ]
```

*Ilustración 50 - Código del fichero urls.py del módulo cysectrainingface*

La *url* de la línea 12 (Ilustración 50) es a la que hay que lanzar la petición HTTPS con las credenciales para obtener los *tokens* de acceso y de refresco. Mientras que la de la línea 13, se lanzará una petición cuando el *token* de acceso haya caducado o se desee cambiarlo. Para ello, es necesario incluir la última clave *"refresh"* obtenida cuando se consulta por los *tokens*.

Hechos estos cambios, el proceso de autenticación está listo para su uso. El siguiente paso tras la consulta de los *tokens*, sería realizar una petición en la que se incluiría la búsqueda deseada.

#### 4.4.4.2. BÚSQUEDA

La búsqueda que se ha pensado para una primera versión de la API, es muy sencilla. Consiste en que el usuario lance una petición HTTPS POST, obviamente siguiendo el proceso indicado en el apartado anterior, indicando en el campo por el cual se desea realizar la búsqueda. Los campos disponibles serán mostrados en la pestaña de "*useful information*" de la web. Una vez el servidor, concretamente la vista programada para tal, reciba la petición, procesará los parámetros de la misma y realizará la búsqueda tomando como criterio la clave y el valor introducido.

```
57  @api_view(['POST'])
58  @permission_classes((IsAuthenticated,))
59  def traininglist(request):
60      with open(os.path.join(BASE_DIR, '../data.json'),'r') as datafile:
61          data = json.load(datafile)
62      keylist = ['word']
63      #Obtiene todas las claves que hay en el fichero json
64      for training in data['resources']:
65          keys = training.keys()
66          for key in keys:
67              if key not in keylist:
68                  keylist.append(key)
69
70      if request.method == 'POST':
71          print(request.POST)
72          parameters = list(request.POST.keys())
73          print(parameters)
74          jsonresult = {}
75          if (len(parameters) == 1):
76                  if parameters[0] in keylist:
77                      print(request.POST.get(parameters[0]))
78                      jsonresult = {"resources": basic_training_search(request.POST.get(parameters[0]),data,parameters[0])}
79                  else:
80                      jsonresult = {"message":"Please, enter a correct parameter name"}
81          else:
82              jsonresult = {"message": "Please, enter only one parameter"}
83
84          return JSONResponse(jsonresult)
```

*Ilustración 51 - Código correspondiente a la función traininglist de la API REST*

En el fragmento de código de la imagen superior, puede observarse como con el decorador ***@api_view(['POST'])*** se está indicando que se aplicará para peticiones *POST*. En cuanto al método "***traininglist(request)***" recibe la consulta por parámetro y procede a cargar los datos del repositorio. Más tarde, en las líneas 64-68 – Ilustración 51, itera sus elementos para obtener toda la variedad de claves presente en el repositorio. A partir de la línea 70 es donde empieza verdaderamente el proceso de búsqueda. Tras comprobar de nuevo que la petición haya sido realizada mediante un POST, recupera la clave con las que se ha realizado la consulta desde los parámetros de la misma y comprueba que haya únicamente una, en caso contrario, devuelve un mensaje de error. Si resulta que la consulta se ha hecho correctamente, hace uso del método visto en la sección 4.4.3.2 búsqueda básica (Ilustración 24), para encontrar aquellos trainings que satisfacen los criterios de la consulta. Por último, añade esos ejercicios a una lista que es devuelta al cliente en formato JSON.

Con esto, ya estaría diseñada e implementada la API REST por lo que el usuario, en este momento, ya puede lanzar consultas contra el servidor. Para ello, deberá apoyarse en los distintos recursos que le proporcione la tecnología con la que desee hacerlo. En este trabajo final de grado, a medida de ejemplo se empleará el comando *curl*.

Para solicitar el *token* de acceso, hay que lanzar el siguiente comando: *curl -k -X POST https://localhost:8000/api/token/ -d username=<username> -d password=<password>*

```
MacBook-Pro-de-Alberto:~ albertocm$ curl -k -X POST https://localhost:8000/
api/token/ -d username="admin" -d password="admin"
{"refresh":"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2tlbl90eXBlIjoicmVmcm
VzaCIsImV4cCI6MTU1OTQ2OTExNiwianRpIjoiYjg3ZGQzODRkNjYxNDlkNGEwMWExYTNlMjM1Z
mI5YmQiLCJ1c2VyX2lkIjo4fQ.2uR5cJIgVmSBef4V4aLkqYlwv7wBlpOf0sAWku6hMc0","acc
ess":"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwi
ZXhwIjoxNTU5Mzg2MzE2LCJqdGkiOiJkOWNjNjQzYzZjNjM0MWQ0ODlhNjQwOGQwNzEyMzk4ZSI
sInVzZXJfaWQiOjh9.aQOx5pWzcG8kchse9EffzMsDm5njW96iqKXfCIU_4kw"}MacBook-Pro-
de-Alberto:~ albertocm$
```

*Ilustración 52 - Comando curl para solicitar tokens JWT*

Para solicitar una búsqueda, se deberá ejecutar el siguiente comando: *curl -k -H "Authorization: Bearer <accesstoken> -X POST https://localhost:8000/trainings/ -d title="Advanced"*

***Nota: es necesario incluir el parámetro -k. Esto se hace para indicar al comando que confíe en el certificado SSL de tal forma que se pueda obtener la respuesta deseada y no una en la que se describe que no se confía en el certificado. Es únicamente con fines demostrativos, puesto que una vez se incluya en producción, con un certificado propio, no será necesario realizar esta operación.***


```
MacBook-Pro-de-Alberto:~ albertocm$ curl -k -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ
0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhwIjoxNTU5Mzg2ODc1LCJqdGkiOiJhMzQ0NGQyMzAxOGI0MTdiYmZhNjg3ODc1OTkzZDlkYSIsInVzZ
XJfaWQiOjh9.HuPETVQ27p4nD6t-3EHUtb6gn0VkW-O1bsQ47GeHWRc" -X POST https://localhost:8000/trainings/ -d title="A
dvanced"
{"resources":[{"id":3,"source":"Enisa","title":"Advanced artefact handling","target_audience":"Technical CERT
staff.","duration":"8 hours","description":"Teach students how to obtain memory images from different sources
and to analyse them. Both Windows and Linux systems will be covered.","files":["Handbook","Toolset","Virtual I
mage 1","Virtual Image 2","AQ Tools"],"urls":["https://www.enisa.europa.eu/topics/trainings-for-cybersecurity-
specialists/online-training-material/documents/advanced-artifact-handling-handbook","https://www.enisa.europa.
eu/topics/trainings-for-cybersecurity-specialists/online-training-material/documents/advanced-artifact-handlin
g-toolset","http://enisa.europa.eu/ftp/styx32.ova","http://enisa.europa.eu/ftp/opsu-12-2-x86.ova","https://www
.enisa.europa.eu/topics/trainings-for-cybersecurity-specialists/online-training-material/files/aq-tools"],"sit
e":"https://www.enisa.europa.eu/topics/trainings-for-cybersecurity-specialists/online-training-material/techni
cal-operational"},{"id":9,"source":"Enisa","title":"Introduction to advanced artefact analysis","target_audien
ce":"CSIRT staff and incident handlers involved in the technical analysis of incidents.","duration":"4 hours",
"description":"This training presents the introduction to the advanced artefact analysis. It is the first part
 of a three-day course introducing assembly language and tools commonly used for the advanced artefact analysi
s.","files":["Handbook","Toolset"],"urls":["https://www.enisa.europa.eu/topics/trainings-for-cybersecurity-spe
cialists/online-training-material/documents/introduction-to-advanced-artefact-analysis.pdf","https://www.enisa
.europa.eu/topics/trainings-for-cybersecurity-specialists/online-training-material/documents/introduction-to-a
dvanced-artefact-analysis-toolset.pdf"],"site":"https://www.enisa.europa.eu/topics/trainings-for-cybersecurity
-specialists/online-training-material/technical-operational"},{"id":25,"source":"Enisa","title":"Advanced Pers
istent Threat incident handling","target_audience":"Incident handlers and technical CERT staff.","duration":"3
 hours","description":"This task provides students with information about methods commonly used by attackers d
uring the Advanced Persistent Threat (APT) attacks as well as methods of discovering and protecting internal r
esources against these attacks. Examples used in the exercise are based on real incidents and observations. Th
e objective is also to involve participants in creative approaches to building CERT capability to deal effecti
vely with and resolve the problem of APT attacks within an organisation.","files":["Handbook","Toolset"],"urls
":["https://www.enisa.europa.eu/topics/trainings-for-cybersecurity-specialists/online-training-material/docume
nts/advanced_persistent_threat_incident_handling_handbook","https://www.enisa.europa.eu/topics/trainings-for-c
ybersecurity-specialists/online-training-material/documents/advanced_persistent_threat_incident_handling_tools
et"],"site":"https://www.enisa.europa.eu/topics/trainings-for-cybersecurity-specialists/online-training-materi
al/operational"}]}MacBook-Pro-de-Alberto:~ albertocm$
```


*Ilustración 53 - Comando curl que solicita una búsqueda*

Con esta secuencia de comandos se puede observar el procedimiento natural de consulta y los resultados obtenidos. Con el primero de ellos, puede verse como se obtienen los 2 *tokens* en formato JSON, mientras que en el segundo puede verse también como tras incluir el *token "Access"* en la cabecera *Authorization* se obtiene la lista de todos los trainings que cumplen el criterio de contener la palabra *Advanced* en su título, de nuevo en formato JSON.

# 5. DESPLIEGUE DEL ENTORNO

En el siguiente punto se va a tratar aquellas cuestiones referentes al despliegue del entorno de desarrollo. Además, se darán una serie de conceptos útiles para llevar a cabo la puesta en producción del sistema, tales como el servidor web a elegir y aspectos a tener en cuenta.

## 5.1. ENTORNO DE DESARROLLO

A lo largo del siguiente apartado se van a exponer aquellas tecnologías que se han empleado para el desarrollo de la solución. Se empezará detallando el lenguaje empleado y sus requerimientos para más tarde entrar en el detalle de Django y su servidor como entorno de desarrollo.

### 5.1.1. PYTHON 3

Python es un lenguaje de programación de código abierto en el que se puede encontrar soporte de orientación a objetos, programación imperativa y programación funcional. Sustituye el uso de caracteres especiales como forma de definir bloques de código a favor de las tabulaciones o espaciados, permitiendo de esta manera un desarrollo más visual. Posee varias versiones, sin embargo, para el desarrollo de este proyecto se ha decidido emplear la versión 3, ya que las anteriores van a dejar de tener soporte a partir de 2020. [30]

En cuanto al manejo de variables otorga mayor versatilidad que otros lenguajes de programación al definirse de manera dinámica. Además, introduce nuevos modelos de tipos de datos como tuplas, similares a los *arrays* pero inmutables, sets, de nuevo como los *arrays* pero sin contenido duplicado y diccionarios, similares a los empleados por JavaScript, en los que se recoge un conjunto de elementos referenciados por clave: valor. [30]

Posee una herramienta de gestión de paquetes denominada "***pip***". Mediante ésta, es posible instalar y administrar paquetes de software escritos en este lenguaje. A lo largo del análisis, diseño e implementación de la solución se han nombrado diversos módulos de Python que favorecían la implantación de distintas funcionalidades. Pues bien, la mayoría de estos, han tenido que ser instalados mediante el uso de *pip*, ejecutando el comando ***pip3 install <nombremodulo>*** en la consola de comandos. [31]

Dicha herramienta posee la habilidad de gestionar los paquetes a través de un fichero, por lo que, si es necesario configurar el entorno de desarrollo en otro host, es una característica realmente útil. A continuación, se describirá aquellos requerimientos relativos a los módulos de este lenguaje, necesarios para el correcto funcionamiento del sistema. Para ello, basta con ejecutar el comando: ***pip freeze > requirements.txt*** de tal forma que la salida del mismo sea enviada a un fichero con ese nombre. Este archivo, podrá encontrarse en la sección de Anexos, concretamente en el **ANEXO III**.

Para configurar otro entorno de desarrollo en una máquina distinta, conviene hacer uso del módulo "***virtualenv***" de Python. Éste, permite crear una especie de contenedor en el que instalar los requerimientos sin que no se produzcan colisiones con otras versiones ya instaladas en el sistema. Se puede lograr siguiendo los siguientes pasos:

1. Lanzar en la consola de comandos: *pip3 install virtualenv*
2. Crear un entorno virtual con el siguiente comando: *virtualenv <nombre_env>*
3. Activarlo mediante el comando: *source <nombre_env>/bin/activate.*
4. Verificar la versión de Python: *python –V*
5. Instalar los requerimientos necesarios: *pip install –r <requeriments.txt>*

Una vez realizado esto, el *host* de desarrollo ya estará listo para empezar a usar el sistema. Tanto los scripts de minado de datos como el proyecto de Django.

### 5.1.2. DJANGO

Anteriormente en el análisis de la solución, ya se ha llevado a cabo una breve explicación acerca de este *web framework*. Sin embargo, se han omitido ciertos aspectos con la intención de dejarlos patentes en este apartado, al considerarlos relevantes al entorno de desarrollo.

Django posee un servidor web destinado al desarrollo, con este, es fácil realizar pruebas del código de manera rápida. El inconveniente que presenta es que no emplea el protocolo HTTPS por lo que no es seguro, cosa normal teniendo en cuenta que no está pensado para su puesta en producción.

Sin embargo, unas de las peticiones era que, pese a que fuese en un entorno de desarrollo, siguiese el protocolo HTTPs. De tal forma que, gracias a la gran comunidad de desarrolladores que posee este *framework*, existe un módulo que permite emplear el mismo servidor bajo este protocolo seguro. Para ello, es necesario instalarlo, como ya se ha visto, mediante el uso de ***pip***. Una vez hecho este paso, podrá ser lanzado de

manera segura, desde el directorio raíz del proyecto, mediante el siguiente comando: ***python3 manage.py runsslserver.*** [32]

## 5.2. CONSIDERACIONES PARA LA PUESTA EN PRODUCCIÓN

A pesar de que no es objetivo de este trabajo la puesta en producción del mismo, a continuación, se van a dar una serie de consideraciones a tener en cuenta para lograr este fin. Así, el objetivo de este apartado es indicar aquellos aspectos a tener en cuenta para un despliegue del sistema en un entorno de producción. Las opciones que se van a exponer no tienen por qué ser tomadas como máximas en un futuro, sino que son el resultado de un proceso de análisis por parte del autor de este documento, que espera resulte de orientación para posibles líneas futuras.

### 5.2.1. ELECCIÓN DE SERVIDOR.

El hecho de escoger un determinado servidor frente a otro es uno de los aspectos más críticos en el despliegue en producción. Dependiendo de las características del sistema, así como del flujo que se espere, resultará mejor opción uno frente a otro.

Los dos principales servidores web sobre los que se puede desplegar un proyecto desarrollado en Django son Apache y NGINX. Ambos requerirán de la instalación de un módulo adicional que convertirá el lenguaje Python llevado a cabo en la implementación de las aplicaciones, para su alojamiento en el servidor.

La diferencia más reseñable entre ambos es su manera de tratar las conexiones. Mientras que en Apache son gestionadas mediante un modelo de hilos, entrando en cola aquellas que sean nuevas. En NGINX son tratadas mediante un modelo asíncrono, basado en arquitectura *event-driven*, permitiendo un mayor número de conexiones simultáneas. [33]

Sin embargo, pese a que en funcionamiento parece ventajoso NGINX, la fácil configuración y flexibilidad de Apache frente a su oponente. Hace que sea la mejor opción si el sitio web no recibe mucho tráfico. Es por esto que, cuando se decida llevar a producción el sistema, habrá que realizar una estimación del tráfico que se recibirá y en función de los resultados, elegir uno u otro. Siendo NGINX el preferible si se piensa que las conexiones a recibir resultarán muy numerosas.

### 5.2.2. NOMBRE DE DOMINIO

El dominio de una web supone una forma textual de traducir la IP en la que se encuentra configurado el servidor web. Es único y ha de elegirse de manera cuidadosa, puesto que significará como el usuario recordará el nombre del sistema e influirá en el posicionamiento de la web en los motores de búsqueda.

Los nombres de dominio se categorizan de diversas formas, siendo los más comunes los geográficos (.es - España, .uk - Reino Unido…), comerciales (.com) o institucionales (.edu - Educación, .org - Organización). Es por esto que será necesario determinar a qué disciplina se quiere que pertenezca el sistema. Teniendo en cuenta su temática, lo apropiado sería .edu o .es. Incluso podría estudiarse incorporarlo como subdominio del subdominio propio de la universidad .uc3m. Una vez se haya decidido su categorización, habrá que acudir al agente registrador correspondiente y verificar que no existe, para su creación.

Disponer de un nombre de dominio es importante para la siguiente cuestión a tener en cuenta: la generación de un certificado SSL.

### 5.2.3. CERTIFICADO SSL

El aspecto de la seguridad ha estado muy presente a lo largo del desarrollo del proyecto. Es por esto que es necesario proporcionar mecanismos que blinden al sistema frente a ataques. Uno de estos es la generación de un certificado SSL que permita conexiones cifradas empleando el protocolo HTTPs, tal y como se ha venido haciendo en el entorno de desarrollo.

La expedición del mismo puede hacerse de múltiples maneras, muchas de las cuales, conllevan pagar por el servicio. Sin embargo, la organización "**Let's encrypt**", proporciona mecanismos para generarlos de manera gratuita. Los únicos requerimientos son disponer de un nombre de dominio y conocer el sistema operativo del servidor en el que se aloja la web. [34]

### 5.2.4. OTROS ASPECTOS REGULATORIOS.

Al desplegar el entorno en producción, será accesible mediante internet, por lo que habrá que tener en cuenta los derechos de los que disponen los usuarios, así como, los

derechos *copyright* de los autores de los materiales contenidos en el repositorio que se publica en la red.

Por un lado, habrá que redactar una **política de privacidad y cookies** cumpliendo con la normativa que se exija. Por otro lado, habrá que estudiar las acciones a tomar para satisfacer los **derechos de autor** de los recursos almacenados en el repositorio.

# 6. PRUEBAS

En esta sección se detallarán todas aquellas pruebas realizadas con el fin de comprobar la robustez del desarrollo software. Como la solución está compuesta por varios módulos se procederá a su clasificación en función de los mismos, pero primeramente se describirá el entorno sobre el que se han desarrollado.

## 6.1. DEFINICIÓN DEL ENTORNO DE PLAN DE PRUEBAS

El desarrollo del código ha sido llevado a cabo prioritariamente sobre un dispositivo de la marca Apple. Concretamente el MacBook Pro 13' 2016 con sistema operativo macOS High Sierra Versión 10.13.6. Disponiendo de un procesador Intel Core i5 a 2GHz y una RAM de 8GB.

Los casos de prueba se van a llevar a cabo sobre la misma infraestructura. A esto hay que añadirle que la versión de Django empleada será la 2.1.7, la misma en la que ha sido codificado. Para más especificaciones de configuración de módulos empleados, acudir al **ANEXO III**.

La plantilla a emplear para mostrar los resultados de los test será la siguiente:

| | CP - XX |
|---|---|
| **Título** | Nombre del test |
| **Módulo** | Módulo de la solución al que pertenece |
| **Objetivo** | Fin con el que se realiza |
| **Entrada** | Valores necesarios |
| **Salida** | Resultado esperado |
| **Evaluación** | Satisfactorio o Error |

*Tabla 4 - Modelo a cumplimentar para cada caso de prueba*

## 6.2. WEB SCRAPERS

Este punto englobará aquellos test realizados a los scripts que se han encargado del minado de datos. Se comprobará sobre todo si el minado de datos se hace manera correcta y si se genera el sistema de directorios correctamente.

| | CP - 01 |
|---|---|
| **Título** | Creación del sistema de directorios "EnisaFiles" |
| **Módulo** | Web scraper |
| **Objetivo** | Comprobar que genera una carpeta con el mismo nombre que el ejercicio |
| **Entrada** | - |
| **Salida** | Sistema de directorios correspondiente a los ejercicios de Enisa |
| **Evaluación** | Satisfactorio |

*Tabla 5 - Caso de prueba 1*

| | CP - 02 |
|---|---|
| **Título** | Creación del sistema de directorios "SeedFiles" |
| **Módulo** | Web scraper |
| **Objetivo** | Comprobar que genera una carpeta con el mismo nombre que el ejercicio |
| **Entrada** | - |
| **Salida** | Sistema de directorios correspondiente a los ejercicios de SEEDLabs |
| **Evaluación** | Satisfactorio |

*Tabla 6 - Caso de prueba 2*

| | CP - 03 |
|---|---|
| **Título** | Generación del fichero JSON |
| **Módulo** | Web scraper |
| **Objetivo** | Comprobar que genera el fichero "data.json" con los ejercicios de ENISA |
| **Entrada** | - |
| **Salida** | Fichero "data.json" con todos los campos rellenos correctamente |
| **Evaluación** | Satisfactorio |

*Tabla 7 - Caso de prueba 3*

| | CP - 04 |
|---|---|
| **Título** | Indexación nuevos elementos en el fichero JSON |
| **Módulo** | Web scraper |
| **Objetivo** | Comprobar que nuevos ejercicios son añadidos sin problemas |
| **Entrada** | Fichero "data.json" |
| **Salida** | Fichero "data.json" ampliado con más elementos |
| **Evaluación** | Satisfactorio |

*Tabla 8 - Caso de prueba 4*

| | CP - 05 |
|---|---|
| **Título** | Extracción de los términos de los documentos |
| **Módulo** | Web scraper |
| **Objetivo** | Comprobar la correcta extracción de los términos |
| **Entrada** | Sistema de ficheros |
| **Salida** | fichero "mainwordfile.txt" |
| **Evaluación** | Satisfactorio |

*Tabla 9 - Caso de prueba 5*

### 6.3. INTERFAZ WEB

En el siguiente apartado se englobarán todas aquellas pruebas en las que se ha querido comprobar el correcto funcionamiento de la interfaz web, tanto por parte del servidor como del navegador web.

| | CP - 06 |
|---|---|
| **Título** | Login |
| **Módulo** | Interfaz web |
| **Objetivo** | Comprobar inicio de sesión correcto para ambos tipos de cuentas |
| **Entrada** | Usuario y Contraseña / Cuenta de Google - Github. |
| **Salida** | Autenticación en el servidor |
| **Evaluación** | Satisfactorio |

*Tabla 10 - Caso de prueba 6*

| | CP - 07 |
|---|---|
| **Título** | Acceso no autorizado |
| **Módulo** | Interfaz web |
| **Objetivo** | Comprobar que el sistema no permite la visualización de los datos si el usuario no está logueado |
| **Entrada** | - |
| **Salida** | Página de error |
| **Evaluación** | Satisfactorio |

*Tabla 11 - Caso de prueba 7*

| | CP - 08 |
|---|---|
| **Título** | Respuesta frente a nuevas fuentes de datos |
| **Módulo** | Interfaz web |
| **Objetivo** | Comprobar que el sistema añade los nuevos campos de las fuentes de datos en la búsqueda avanzada y en las notas sobre la búsqueda. |
| **Entrada** | Fichero JSON con nuevos datos provenientes de otra fuente |
| **Salida** | Campos correctos en checkboxes, selects y en las notas sobre búsqueda. |
| **Evaluación** | Satisfactorio |

*Tabla 12 - Caso de prueba 8*

| | CP - 09 |
|---|---|
| **Título** | Búsqueda básica |
| **Módulo** | Interfaz web |
| **Objetivo** | Comprobar la respuesta del sistema frente a una búsqueda sencilla. |
| **Entrada** | Criterios de búsqueda |
| **Salida** | Resultado de la búsqueda esperado |
| **Evaluación** | Satisfactorio |

*Tabla 13 - Caso de prueba 9*

| | CP - 10 |
|---|---|
| **Título** | Búsqueda avanzada extensa (más de 6 criterios) |
| **Módulo** | Interfaz web |
| **Objetivo** | Comprobar la respuesta del sistema frente a largas búsquedas avanzadas. |
| **Entrada** | Criterios de búsqueda |
| **Salida** | Resultado de la búsqueda esperado |
| **Evaluación** | Satisfactorio |

*Tabla 14 - Caso de prueba 10*

| | CP - 11 |
|---|---|
| **Título** | Cerrar sesión de usuario |
| **Módulo** | Interfaz web |
| **Objetivo** | Comprobar que la sesión del usuario se cierra correctamente. |
| **Entrada** | - |
| **Salida** | - |
| **Evaluación** | Satisfactorio |

*Tabla 15 - Caso de prueba 11*

## 6.4. API REST

En el siguiente punto se expondrán aquellas pruebas que se han realizado conforme a las funcionalidades de la API REST.

| | CP - 12 |
|---|---|
| **Título** | Request sin authorization token |
| **Módulo** | API REST |
| **Objetivo** | Comprobar que el sistema requiere la cabecera authorization |
| **Entrada** | Petición HTTPS sin cabecera Authorization |
| **Salida** | Solicitar credenciales al usuario |
| **Evaluación** | Satisfactorio |

*Tabla 16 - Caso de prueba 12*

| | CP - 13 |
|---|---|
| **Título** | Request con un parámetro que no existe |
| **Módulo** | API REST |
| **Objetivo** | Comprobar que el sistema posee mecanismos que avisan del error |
| **Entrada** | Petición HTTPS con un nombre de parámetro erróneo |
| **Salida** | Mensaje de información |
| **Evaluación** | Satisfactorio |

*Tabla 17 - Caso de prueba 13*

| | CP - 14 |
|---|---|
| **Título** | Request de búsqueda sencilla |
| **Módulo** | API REST |
| **Objetivo** | Comprobar que el sistema procesa bien las búsquedas |
| **Entrada** | Petición HTTPS |
| **Salida** | Lista de ejercicios que coincidan con la búsqueda |
| **Evaluación** | Satisfactorio |

*Tabla 18 - Caso de prueba 14*

# 7. GESTIÓN DEL PROYECTO

A lo largo del siguiente punto se expondrán los detalles de la planificación y gestión de este trabajo de fin de grado. Para ello, primeramente, se expondrá la herramienta que se ha empleado para la administración del software, para más tarde, describir los recursos consumidos para la elaboración del mismo.

## 7.1. GITHUB COMO REPOSITORIO ONLINE

El desarrollo correspondiente a este proyecto, se ha llevado a cabo de la mano de una herramienta de gestión de software: GIT. Consiste en un sistema de control de versiones distribuido, diseñado para manejar desde pequeños a grandes proyectos con eficiencia y rapidez. [35]

Concretamente se ha empleado el propio de la compañía GitHub, Inc. Este permite la visualización de los cambios en una interfaz web facilitando la administración de los mismos. Además, otorga control sobre accesos, de tal forma que el usuario puede definir quién puede ejercer cambios en el proyecto.

Las ventajas que supone emplear este tipo de herramientas es que se puede garantizar una recuperación frente a posibles pérdidas, al tratarse de un repositorio online. Además, permite regresar a una versión anterior si los cambios hechos no satisfacen las necesidades.

El funcionamiento concreto de este tipo de tecnología consiste en la existencia de un repositorio local compuesto por **un directorio de trabajo** donde se encuentran los archivos editables, el ***index,*** que indica el estado de los mismos y por último el ***head*** que apunta al último cambio realizado. De tal forma que, cuando se modifica un fichero, basta con lanzar los comandos correspondientes al registro de los cambios: ***git add <filename>***, lo que provocará un registro en los cambios en el *index* y ***git commit –m "message"*** para incluir los cambios del *index* al *head*. Por último, para enviar los cambios de la copia local al repositorio online bastaría con ejecutar el siguiente comando: **git push** y de esta forma se guardarían todos los cambios realizados teniendo así una copia de seguridad a la vez que un control de versiones.

Cabe destacar que para su uso se ha empleado una cuenta *premium*, lo que permite mantener los desarrollos en privado. Este es el caso del proyecto al que nos referimos, de tal forma que, nadie distinto al autor del mismo, ha colaborado con la elaboración o ha visualizado el código.

## 7.2. PLANIFICACIÓN

La gestión del tiempo se ha llevado a cabo mediante un diagrama de Gantt. Para la realización del mismo, ha sido necesario determinar la duración total del proyecto. Éste tuvo su punto de partida en el 18 de enero de 2019 y su final será, con la presentación del mismo en la primera quincena del mes de julio de 2019.

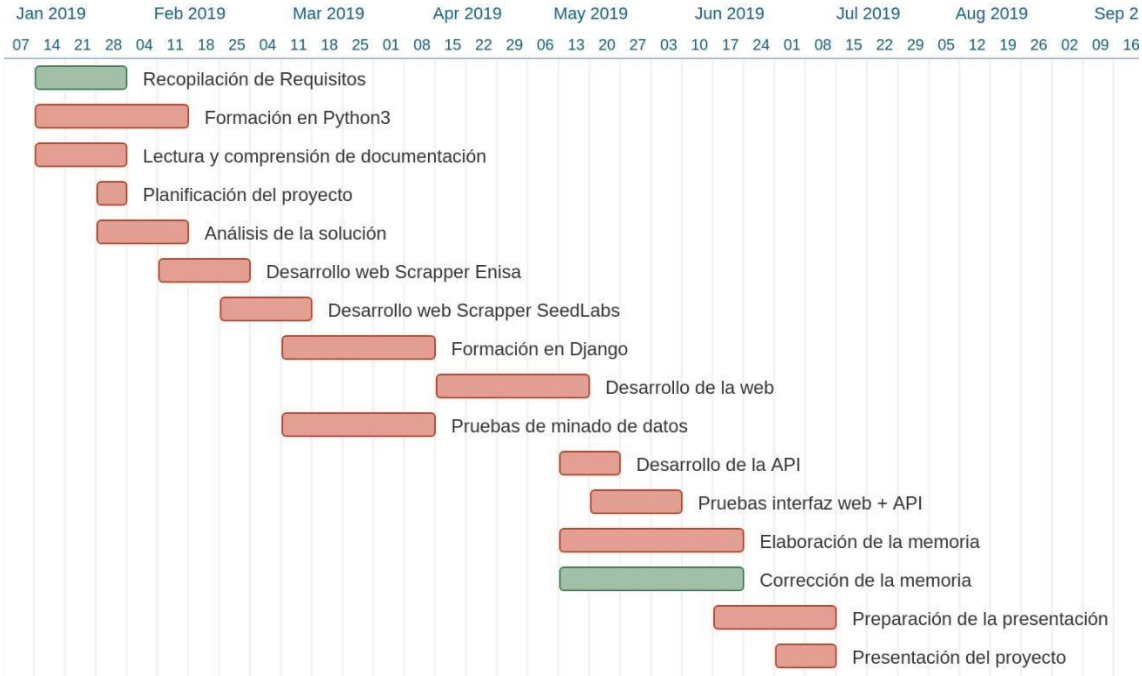



*Ilustración 54 - Diagrama de gantt del proyecto*

Tal y como puede observarse, se detectaron 16 tareas de las cuales 14 fueron asignadas al alumno a cargo del proyecto (en rojo) y 2 a su tutora (en verde). Puede apreciarse, que el grueso de las horas dedicadas se ha empleado en el desarrollo de los distintos módulos de la solución. Además, se han destinado suficientes horas a la formación en las tecnologías empleadas, Python y Django, puesto que el desarrollador no disponía de conocimientos previos sobre ellas, ambas han requerido aproximadamente, un mes de duración, aunque se han podido compaginar con otras actividades. Estas tareas mencionadas, han sido sin duda, las más críticas, puesto que un retraso frente a su estimación, generaría un desajuste en el grueso del proyecto.

Debido a la situación personal del alumno, en la que ha tenido que compaginar trabajo con el desarrollo del proyecto, se ha estimado que dedicará al día 2.5 horas para la resolución del mismo, lo que sumará un total de 375 horas, teniendo en cuenta fines de semana. Cifra superior al total de horas exigidas en la ficha de la asignatura: 204 h. [36]. Es necesario destacar que seguir la planificación al 100% es prácticamente imposible, es

por esto que se ha ideado recuperar aquellas horas que no se hayan podido invertir durante la semana laboral, en los fines de semana.

La tutora, por su parte, llevará a cabo labores de recopilación de requisitos y correcciones, cuyo tiempo estimado en total sumará 6 horas de la primera actividad más 20 de la segunda, un total de 26 horas.

## 7.3. PRESUPUESTO ESTIMADO

En el siguiente punto se hará una estimación del coste que ha supuesto el desarrollo del proyecto. Es necesario destacar que, al estar compuesto por software los costes vendrán derivados de la escasa amortización que se produzca del uso del hardware en el periodo que dure el proyecto junto a los costes personales del mismo.

Es cierto que, al tratarse de un desarrollo web, podría implicar disponer de costes de infraestructura, es decir, aquellos dedicados al mantenimiento y operaciones del servidor en el que se alojase. Sin embargo, al no poner el sistema en producción no tenemos que tenerlos en cuenta de momento.

El perfil del alumno es el correspondiente a un estudiante de último curso de la Universidad Carlos III de Madrid con aproximadamente 6 meses de experiencia laboral relacionada con el desarrollo. Por tanto, según un estudio realizado por la empresa de recursos humanos HAYS, el sueldo medio de los desarrolladores de Python en la ciudad de Madrid ronda los 34.000 euros de media cuando la experiencia es de 0 a 2 años. [37]

Teniendo en cuenta estos números, es fácilmente calculable el precio por hora de este recurso 34.000 € al año entre 260 días laborales y 8 horas de jornada asciende a 18,47 € la hora.

En cuanto al sueldo correspondiente a las labores llevadas a cabo por el docente, se ha tenido que hacer una tarea de estimación en función de las retribuciones publicadas por la propia universidad para 2018 [41]. Existen numerosos parámetros por los cuales se mide su salario, sin embargo, al desconocerse el valor de alguno de ellos, y por considerar que tampoco es apropiado obtenerlo detalladamente, se procederá a hacer una media de ellos.

| | Cuantía (€/mes) | Conocido/Estimado | Total |
|---|---|---|---|
| **Sueldo base** | 1148,34 | | 1148,34 |
| **Complemento de destino** | 824,03 | Máximo entre 2 | 412,015 |
| **Complemento específico y específico adicional** | 473,24 | Máximo entre 2 | 236,62 |
| **Trienios** | 44,18 | 20 años trabajados | 265,08 |
| **Complemento específico por méritos docentes** | | Media de los 3 niveles | 127,97 |
| Nivel 29 | 153,84 | | |
| Nivel 27 | 124,61 | | |
| Nivel 26 | 105,45 | | |
| **Complemento productividad investigadora** | | Media de los 3 niveles | 127,97 |
| Nivel 29 | 153,84 | | |
| Nivel 27 | 124,61 | | |
| Nivel 26 | 105,45 | | |
| | | | 2317,99 |

*Tabla 19 - Presupuesto estimado de los honorarios del docente*

De estos 2318 €, dividido entre 23 días de media de trabajo al mes, se obtiene unos honorarios diarios de 100,78 € que, repartidos entre las 8 horas de la jornada laboral, se obtiene un precio de 12,59 € la hora. Con estos cálculos hechos y conociendo las horas estimadas que se le han imputado, es fácil calcular el coste total del mismo

| Tarea | Horas dedicadas | €/hora | €/Tarea |
|---|---|---|---|
| Recopilación de Requisitos | 6 | 12,59 | 75,54 |
| Formación en Python 3 | 23 | 18,47 | 424,81 |
| Lectura y Comprensión de documentación | 20 | 18,47 | 369,4 |
| Planificación del proyecto | 6 | 18,47 | 110,82 |
| Análisis de la solución | 30 | 18,47 | 554,1 |
| Desarrollo *web scrapper* Enisa | 22,5 | 18,47 | 415,575 |
| Desarrollo *web scrapper* SeedLabs | 22,5 | 18,47 | 415,575 |
| Formación en Django | 30 | 18,47 | 554,1 |
| Desarrollo de la web | 45 | 18,47 | 831,15 |
| Desarrollo de la API | 20 | 18,47 | 369,4 |
| Pruebas de desarrollo | 15 | 18,47 | 277,05 |
| Elaboración de la memoria | 135 | 18,47 | 2493,45 |
| Corrección de la memoria | 20 | 12,59 | 251,8 |
| Preparación de la presentación | 6 | 18,47 | 110,82 |
| | | | 7253,59 |

*Tabla 20 - Costes personales del proyecto*

Se puede ver como el grueso del proyecto se ha llevado a cabo en labores de desarrollo y elaboración de la memoria. Empleando el 31,25% y 33,75% respectivamente, del total de horas imputadas. El 13,25% de las horas han sido destinadas a formación en Python y Django. Por último, el porcentaje de tiempo restante se ha invertido en otras tareas referentes a la comprensión, análisis y preparación del proyecto. Lo que ha arrojado un total de **7223,59€** según las estimaciones por una carga de trabajo correspondiente a unas **401 horas**.

A estos costes personales, hay que añadirle los costes derivados del uso de material. Dentro de esta categoría se encontrará la amortización sufrida por el portátil en el que se ha llevado a cabo el desarrollo, así como diversos materiales de oficina que han servido de apoyo.

| Bien | Coste unitario (€) | Amortización (€) | Coste imputable (€) |
|---|---|---|---|
| Portátil Macbook pro 13" 2016 | 1500 | 150 | 150 |
| Material de oficina | 30 | - | 30 |
| | | | 180 |

*Tabla 21 - Costes materiales del proyecto*

Para el cálculo de la amortización se ha seguido la siguiente fórmula:

$$\frac{nº\ Meses\ de\ utilización\ del\ equipo}{Periodo\ de\ depreciación\ (60\ meses)} \times\ Coste\ \times\ Asignación\ al\ proyecto\ (\%)$$

Por lo que teniendo una asignación del 100% durante 6 meses y con ese precio unitario se obtiene una amortización de 150€.

Por último, para conocer el valor total del proyecto solo falta obtener los costes directos mediante la suma de los costes personales y los materiales. Una vez calculado, se le aplicará un porcentaje del 10% que constituirán los costes indirectos asociados al desarrollo del mismo. Con la suma de los costes directos e indirectos, se le asociará un beneficio del 25% de estos, que simbolizará las ganancias. Para terminar, el IVA será incluido en el presupuesto.

| | |
|---|---|
| **Costes de personal (€)** | 7253,59 |
| **Costes de materiales (€)** | 180 |
| **Total costes directos (€)** | 7433,59 |
| **Total costes indirectos (10% directos) (€)** | 743,36 |
| **Total costes (directos + indirectos) (€)** | 8176,95 |
| **Beneficio (25%) (€)** | 2044,24 |
| **Total, sin IVA (€)** | 10221,19 |
| **Total, con IVA (21%) (€)** | **12367,64** |

*Tabla 22 - Coste total del proyecto*

Es entonces cuando se puede afirmar que el presupuesto estimado total para el desarrollo de este proyecto ascenderá a **12367,64 €.**

# 8. MARCO REGULATORIO Y NORMATIVA.

A lo largo de la siguiente sección se incluirán todos aquellos aspectos referentes a la legalidad y normativa vigente que afecten al desarrollo de este Trabajo de Fin de Grado. Por un lado, se explicarán aquellos conceptos de la ley de protección de datos que repercutan en el sistema, así como todas aquellas consideraciones de *copyright* o uso de los datos que haya que tener en cuenta.

## 8.1. GDPR

Han sido muchos los años en los que cada país de la unión europea regía su normativa propia en cuanto a la ley de protección de datos. En España, disponíamos de la Ley Orgánica de Protección de Datos (LOPD) mediante la cual únicamente eran protegidos los ciudadanos españoles y sólo aquellos datos que fuesen personales. Además, no era obligatorio comunicar brechas de seguridad.

Sin embargo, el 25 de mayo de 2018 entró en vigor el Reglamento Europeo de Protección de Datos, lo cual obligó a todas las empresas de la unión europea a regirse bajo la misma regulación en lo referente al tratamiento de los datos. En primer lugar, los ciudadanos europeos son todos protegidos de la misma manera, de tal forma que, en el caso de España, ya no sólo se protegen los datos personales, sino que también los genéticos y datos biométricos. En segundo lugar, las comunicaciones de brechas de seguridad pasan a ser obligatorias en un plazo inferior a 72 horas.

En cuanto a los derechos de los usuarios, la recogida de sus datos ha de ser concisa, transparente, inteligible y con un lenguaje claro y sencillo. Teniendo derecho también al olvido o portabilidad de sus datos.

Por último, con esta nueva regulación, surge la figura del responsable del tratamiento de los datos que llevará a cabo labores acordes a la definición de su cargo. Obliga, a la verificación, evaluación y valoración regulares de las medidas adoptadas. [39]

GDPR es, por tanto, un factor a tener en cuenta en el marco regulatorio en el que se posiciona este proyecto, ya que al hacer un tratamiento de los datos (usuario y credenciales) ha de estar lo suficientemente protegido para cumplir con dicha regulación.

## 8.2. DERECHOS DE AUTOR

Constituyen el conjunto de normas que garantiza los derechos morales de los autores de obras publicadas o inéditas. Son uno de los derechos fundamentales presente en la Declaración Universal de los Derechos Humanos.

La Unión Europea posee su propio organismo regulador que actúa para proteger estos derechos. Distingue entre propiedad industrial, aquella referente al desarrollo de patentes, modelos industriales, marcas..., y los derechos de autor. [40]

Existen diversas clases de derechos de autor, pero en el sistema implementado a lo largo de este trabajo, aquel que afecta es el referente a los derechos de explotación de los materiales docentes. Esta clase, permite al autor de los mismos evitar la copia, reproducción, distribución, transformación y comunicación pública de la obra a terceros. Por lo tanto, cualquiera de estas acciones ha de estar permitida por el autor de la obra y de no estar explícitamente indicado habrá que contactar con el autor de la misma. [41]

En cuanto a las fuentes de datos empleadas en este trabajo, ambas permiten la reproducción y distribución en cualquier medio del material siempre y cuando no se emplee con fines comerciales.



*Ilustración 55 - Copyright de los ejercicios de SEEDLabs [7]*



*Ilustración 56 - Copyright de los ejercicios de ENISA [6]*

## 8.3. SOBRE EL MINADO DE DATOS - ROBOTS.TXT

Se trata de un fichero de texto que incluye directrices a seguir para los robots araña o *web scrapers*. Su función es meramente consultiva por lo que no podrán bloquear las consultas que estos hagan en caso de no querer que así sea. Sin embargo, dadas las circunstancias académicas de este trabajo, se ha concluido que ha de seguirse las normas que establezcan.

Se pueden encontrar añadiendo la extensión */robots.txt* al final de la url de la web. De esta forma se puede consultar si la página permite el acceso a este tipo de robots o no. En el caso de ENISA, se han encontrado las líneas de la imagen inferior, mientras que para SEEDLabs no se ha podido acceder al mismo.

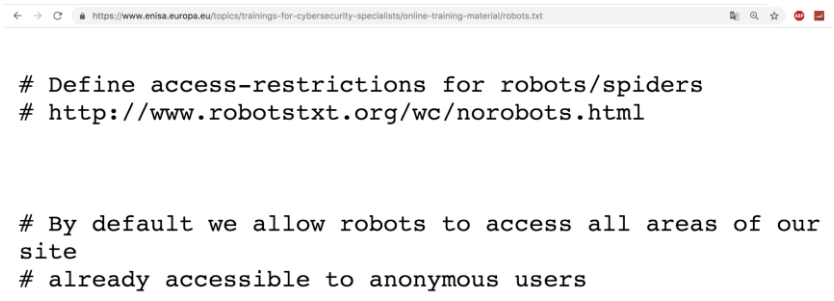



*Ilustración 57 - Extensión robots.txt de la web de ENISA [6]*

# 9. CONCLUSIONES Y LÍNEAS FUTURAS.

En el siguiente punto, se van a tratar aquellas conclusiones a las que se han llegado a lo largo del desarrollo de este Trabajo Fin de Grado, relativas al cumplimiento de los objetivos, así como personales. Además, se incluirán una serie de líneas futuras mediante las cuales el alcance del proyecto podrá ser ampliado.

## 9.1. CONSECUCIÓN DE OBJETIVOS

El cumplimiento de los objetivos expuestos ha guiado el desarrollo del software en todo momento. Sin embargo, antes de comenzar con el análisis referente a si se han logrado, es necesario indicar un par de aspectos referentes a la memoria. Por un lado, el hecho de que este trabajo suponga el comienzo de un proyecto más extenso, así como la necesidad de que sea escalable, ha originado que el estudio de la solución se haya tenido que hacer de forma detallada, para tratar de evitar los máximos imprevistos posibles en el futuro. Por otro lado, en cuanto a la redacción del diseño e implementación de la solución, tal y como se habrá podido observar, se ha detallado lo máximo posible la implementación del código, con el objeto de facilitar el entendimiento del mismo a futuros estudiantes que se encarguen de las líneas futuras.

El objetivo relacionado con el módulo de la solución de elaboración del conjunto de datos, ha requerido de un análisis exhaustivo de cual podría llegar a ser la mejor arquitectura para su almacenamiento y obtención. Una vez concluido en dicho análisis, que la mejor opción es codificar *web scrapers* y almacenar los datos en un sistema de ficheros, y habiéndolo implementado, puede afirmarse con rotundidad que ha sido un acierto, puesto que se ha conseguido que se haga de manera automática, es escalable y, sobre todo, sencillo.

En cuanto a los referentes a la implementación de interfaces de acceso, tanto para humanos, como para desarrolladores, se puede verificar fácilmente en las pruebas que los resultados obtenidos son satisfactorios. Además, tal y como se ha podido ver en el apartado de diseño e implementación, se ha llevado a cabo de tal forma que, en el caso de añadir nuevas fuentes de datos, en un principio, no haya que agregar más líneas de código, por lo que se ha cumplido con que sea escalable y con el objetivo de no tener que volver a programar una lógica ya implementada.

Por último, en cuanto a los aspectos de seguridad, al tratarse de un trabajo relacionado con esta área no se podía dejar de lado. Para la consecución de este objetivo, se ha hecho uso de todas las facilidades que otorga Django como *framework*. Se ha podido

ver que proporciona bastantes mecanismos que blindan el sistema frente a ataques. De los empleados cabe mencionar la generación del *token csfr* contra los ataques *cross-site request forgery*, o aquellos mecanismos empleados para la autenticación de los usuarios, tanto en la interfaz web como en la API.

En conclusión, se puede considerar que los objetivos detallados al comienzo del trabajo han sido cumplidos con creces, por lo que se puede estar satisfecho con la realización del mismo.

## 9.2. CONCLUSIONES PERSONALES

Ruego que en este apartado se me perdone el uso de la primera persona y el abandono de los tiempos impersonales.

La elección de un TFG no es una labor trivial, ha de ser lo suficientemente atractivo para enganchar al que lo realiza, además de suponer un reto, puesto que su fin es demostrar todo lo aprendido a lo largo del grado universitario. Con esto en mente, supe que quería realizarlo sobre algún aspecto relacionado con la ciberseguridad, puesto que siempre ha sido un sector de mi interés.

Tras un breve estudio de los profesores que podrían proporcionarme un trabajo con esta temática, contacté con Ana Isabel, la tutora. Ella me propuso la realización del proyecto que nos acontece e inmediatamente acepté, ya que reunía ambos factores que me apasionan y de los que quería aprender y formarme: la ciberseguridad y la programación web.

Además, a este hecho se le sumaba que la tecnología que me sugirió para su desarrollo, era totalmente desconocida para mí, por lo que, si el tema ya había captado mi interés, se acababa de convertir en un reto también: aprender Python.

Ahora que he completado la realización del mismo, puedo concluir que ha sido una decisión muy acertada. El hecho de elegir un tema acorde a mis gustos ha hecho que no me resulte duro dedicarle horas. Además, desarrollarlo desde cero, sin saber programar en Python y por lo tanto desconocer el funcionamiento de Django, ha supuesto que sea un proceso de aprendizaje muy bonito y con unos resultados bastante satisfactorios, lo cual siempre es bueno.

Por último, el hecho de que el trabajo suponga el punto de partida para muchos otros, aparte del cuidado que me ha forzado a tener y del cual he aprendido bastante, hace

que me sienta realizado y feliz por poder contribuir a esta universidad que tanto ha aportado en mi formación.

## 9.3. LÍNEAS FUTURAS

Este **T**rabajo de **F**in de **G**rado supone el punto de partida de un proyecto mucho más ambicioso. Es por esto que las futuras líneas son muy amplias y diversas. De todas las posibles, en este documento se van a recoger aquellas que atañen más directamente al mismo.

- **Gestión avanzada de usuarios:** añadir funcionalidades adicionales a la administración de usuarios. Como puede ser la automatización de la generación de la contraseña o mecanismos para la gestión de la misma.

- **Ampliación de la funcionalidad de la API:** implementar el alcance de la búsqueda de tal forma que permita realizar una avanzada, al igual que la de la interfaz web.

- **Puesta en producción del entorno:** configurarlo en un servidor con dominio propio y con certificado SSL que permita la comunicación segura mediante el protocolo HTTPs.

- **Analizar el resto de recursos:** al igual que se ha hecho con los documentos PDF, llevar a cabo un análisis del resto de recursos que conforman los ejercicios. Entre los que destacan las imágenes de las máquinas virtuales y los distintos *scripts* que proporcionan.

- **Añadir nuevas fuentes de datos:** buscar e implementar el minado de información de otras webs o recursos de la red. Explotando de esta manera, el potencial escalable implementado en este proyecto.

- **Añadir funcionalidades al usuario:** como, por ejemplo, permitir guardar en una lista individual ejercicios de su interés, o un historial de las búsquedas realizadas.

- **Estudiar la migración del sistema a una base de datos no relacional:** tener presente a lo largo de la vida útil del sistema sus prestaciones y consumo de recursos, para plantear una migración a dicho sistema de almacenamiento, más acorde a un tratamiento de datos más activo y masivo.

# 10. BIBLIOGRAFÍA


[1]"MAPA | Mapa en tiempo real de amenazas cibernéticas Kaspersky", MAPA | Mapa en tiempo real de amenazas cibernéticas Kaspersky, 2019. [En línea]. Disponible en: https://cybermap.kaspersky.com/es. [Accedido: 19- May- 2019].

[2] "(ISC)² Global shortfall of cybersecurity workers to reach 1.8 million in five years, new research reveals", International Information Systems Security Certification Consortium, 2017 [En línea]. Disponible en: https://www.isc2.org/~/link.aspx?_id=B562D599FA1448D881F28DD3115A416F&_z=z [Accedido: 11-May2019].

[3] "Másteres y grados en ciberseguridad en España.", Incibe.es, 2019. [En línea]. Disponible en: https://bit.ly/2GdqJUL [Accedido: 25- May- 2019].

[4] "Instituciones que imparten formación en ciberseguridad en España", Incibe.es, 2019. [En línea]. Disponible en: https://bit.ly/2JZS0O2 [Accedido: 25- May- 2019].

[5] "Cybersecurity Certifications | National Initiative for Cybersecurity Careers and Studies", Niccs.us-cert.gov, 2019. [En línea]. Disponible en: https://niccs.us-cert.gov/featured-stories/cybersecurity-certifications. [Accedido: 25- May- 2019].

[6] "ENISA", Enisa.europa.eu, 2019. [En línea]. Disponible en: https://www.enisa.europa.eu/. [Accedido: 25- May- 2019].

[7] "SEED Project", Cis.syr.edu, 2019. [En línea]. Disponible en: http://www.cis.syr.edu/~wedu/seed/Labs_16.04/. [Accedido: 25- May- 2019].

[8] "IEEE 1484.12.1-2002 - IEEE Standard for Learning Object Metadata", Standards.ieee.org, 2002. [En línea]. Disponible en: https://standards.ieee.org/standard/1484_12_1-2002.html. [Accedido: 14- Jun- 2019].

[9] "MERLOT", Merlot.org, 2019. [En línea]. Disponible en: https://www.merlot.org/merlot/ [Accedido: 08- Jun- 2019].

[10] "Home", ARIADNE PLUS, 2019. [En línea]. Disponible en: https://ariadne-infrastructure.eu/. [Accedido: 08- Jun- 2019].

[11] "Datos enlazados", Es.wikipedia.org, 2019. [En línea]. Disponible en: https://es.wikipedia.org/wiki/Datos_enlazados. [Accedido: 08- Jun- 2019].

[12] C. Keßler and T. Kauppinen, Linked Open Data University of Münster – Infrastructure and Applications. Münster, 2012.

[13] Y. Ma, B. Xu, Y. Bai and Z. Li, Building Linked Open University Data - Tsinghua University Open Data as a showcase. Pekín, 2012.

[14] "CTF: Entrenamiento en seguridad informática", INCIBE-CERT, 2014. [En línea]. Disponible en: https://www.incibe-cert.es/blog/ctf-entrenamiento-seguridad-informatica. [Accedido: 14- Jun- 2019].

[15] "What is a Cyber Range? - Definition from Techopedia", Techopedia.com. [En línea]. Disponible en: https://www.techopedia.com/definition/28613/cyber-range. [Accedido: 14- Jun- 2019].

[16] "What is Cyber Defense? - Definition from Techopedia", Techopedia.com. [En línea]. Disponible en: https://www.techopedia.com/definition/6705/cyber-defense. [Accedido: 14- Jun- 2019].

[17] "Web Scraping", Wikipedia, 2018 [En línea]. Disponible en: https://es.wikipedia.org/wiki/Web_scraping [Accedido: 18-May2019]

[18] "Bases de datos NoSQL. Qué son y tipos que nos podemos encontrar.", Acens, 2014 [En línea]. Disponible en: https://www.acens.com/wp-content/images/2014/02/bbdd-nosql-wp-acens.pdf [Accedido: 17-May2019]

[19] "Django overview | Django", *Djangoproject.com*, 2019. [En línea]. Disponible en: https://www.djangoproject.com/start/overview/. [Accedido: 25- May- 2019].

[20] "API REST: qué es y cuáles son sus ventajas en el desarrollo de proyectos", BBVAOpen4U, 2019. [En línea]. Disponible en: https://bbvaopen4u.com/es/actualidad/api-rest-que-es-y-cuales-son-sus-ventajas-en-el-desarrollo-de-proyectos. [Accedido: 01- Jun- 2019].

[21] "JSON", Json.org, 2019. [En línea]. Disponible en: https://www.json.org/. [Accedido: 26- May- 2019].

[22] "wget", Bitbucket.org, 2015. [En línea]. Disponible en: https://bitbucket.org/techtonik/python-wget/src/default/. [Accedido: 25- May- 2019].

[23] "Beautiful Soup Documentation — Beautiful Soup 4.4.0 documentation", Crummy.com, 2015. [En línea]. Disponible en: https://www.crummy.com/software/BeautifulSoup/bs4/doc/. [Accedido: 25- May- 2019].

[24] R. Qaiser, "How to Extract Words from PDFs with Python", Medium, 2017. [En línea]. Disponible en: https://medium.com/@rqaiserr/how-to-convert-pdfs-into-searchable-key-words-with-python-85aab86c544f. [Accedido: 27- May- 2019].

[25] V. Freitas, "How to Add Social Login to Django", Simple is Better Than Complex, 2016. [En línea]. Disponible en: https://simpleisbetterthancomplex.com/tutorial/2016/10/24/how-to-add-social-login-to-django.html. [Accedido: 28- May- 2019].

[26] "Documentación de Django | Documentación de Django | Django", Docs.djangoproject.com, 2019. [En línea]. Disponible en: https://docs.djangoproject.com/es/2.2/ [Accedido: 29- May- 2019].

[27] "Fosstack Blog", Fosstack.com, 2017. [En línea]. Disponible en: https://fosstack.com/how-to-add-google-authentication-in-django/. [Accedido: 29- May- 2019].

[28] "JSON Web Token", Es.wikipedia.org, 2019. [En línea]. Disponible en: https://es.wikipedia.org/wiki/JSON_Web_Token. [Accedido: 30- May- 2019].

[29] "davesque/django-rest-framework-simplejwt", GitHub, 2019. [En línea]. Disponible en: https://github.com/davesque/django-rest-framework-simplejwt. [Accedido: 30- May- 2019].

[30] "Python", Es.wikipedia.org, 2019. [En línea]. Disponible en: https://es.wikipedia.org/wiki/Python. [Accedido: 03- Jun- 2019].

[31] "Pip (administrador de paquetes)", Es.wikipedia.org, 2018. [En línea]. Disponible en: https://es.wikipedia.org/wiki/Pip_(administrador_de_paquetes). [Accedido: 03- Jun- 2019].

[32] "teddziuba/django-sslserver", GitHub, 2019. [En línea]. Disponible en: https://github.com/teddziuba/django-sslserver. [Accedido: 03- Jun- 2019].

[33] "NGINX vs. Apache (Pro/Con Review, Uses, & Hosting for Each) - HostingAdvice.com", HostingAdvice.com. [En línea]. Disponible en: https://www.hostingadvice.com/how-to/nginx-vs-apache/. [Accedido: 16- Jun- 2019].

[34] "Getting Started - Let's Encrypt - Free SSL/TLS Certificates", Letsencrypt.org, 2019. [En línea]. Disponible en: https://letsencrypt.org/getting-started/. [Accedido: 07- Jun- 2019].

[35] “Git” [En línea]. Disponible en: https://git-scm.com/ [Accedido: 22-May-2019]

[36] Www3.uc3m.es, 2019. [En línea]. Disponible en: http://www3.uc3m.es/reina/CRONOGRAMAS/Idioma_1/2018/215.13435.pdf?time=1559561577790. [Accedido: 03- Jun- 2019].

[37] "Calculadora Salarial | Trabajadores: Guía del Mercado Laboral 2019 - Hays", Guiasalarial.hays.es, 2019. [En línea]. Disponible en: http://guiasalarial.hays.es/trabajador/calculadora-salarial. [Accedido: 03- Jun- 2019].

[38] "Retribuciones | UC3M", Uc3m.es, 2018. [En línea]. Disponible en: https://www.uc3m.es/conocenos/recursos-humanos/retribuciones. [Accedido: 03- Jun- 2019].

[39] "LOPD vs GDPR principales cambios a tener en cuenta | Cad&Lan", CAD&LAN, 2019. [En línea]. Disponible en: https://www.cadlan.com/noticias/lopd-vs-gdpr-principales-cambios-a-tener-en-cuenta/. [Accedido: 09- Jun- 2019].

[40] "Derecho de autor", Es.wikipedia.org, 2019. [En línea]. Disponible en: https://es.wikipedia.org/wiki/Derecho_de_autor. [Accedido: 09- Jun- 2019].

[41] "GUÍA DE DERECHOS DE AUTOR PARA MATERIALES UTILIZADOS EN AULA GLOBAL", Aulaglobal.uc3m.es, 2017. [En línea]. Disponible en: https://aulaglobal.uc3m.es/soporte/Guia_de_Derechos_de_autor_para_materiales_docentes_en_Aula_Global.pdf. [Accedido: 09- Jun- 2019].

# ANEXO I - ABSTRACT

## INTRODUCTION

Nowadays, there are a lot of platforms where you can find educational resources. Some of them, create and publish their own trainings, but others, just take information from a common database to facilitate the search to the person who wants to know more about a specific topic.

The cybersecurity sector is having an unexpected growth in the last years. The number of network attacks has increased a lot, and this area has neither enough, nor qualified experts. This fact has induced to the occurrence of a huge number of new organizations, university degrees and masters which look for filling this gap.

Under this context, the idea of creating a dataset of exercises and training material with an access interface and an infrastructure according to its needs came up. The objective thoughts were wide and several. The most important was that the system had to be granular and scalable in order to prepare the solution for more data sources. Moreover, the training contents had to be analyzed, and a graphical and development interface needed to be provided.

On the following abstract, several points are going to be treated. Starting with a brief introduction already explained, passing through some relevant aspects of the state-of-the-art and the solution analysis until the design, implementation and deploy phases, where it will be detailed all the techniques and technologies involved. To finish up with the information relevant to tests, project management, regulatory framework and the conclusions.

## STATE OF THE ART

On this section, some points regarding to the different solutions present on the market will be treated. It is necessary to say that currently, there are a lot of educational materials all over the web. Because of this fact, learning object repositories have emerged. They consist in a kind of digital library where the educators can share, manage and reuse the educational resources.

Around this concept, some organizations have been working with the target of getting the largest number of educational materials in order to implement a wide range of interest solutions. It is easy to find online repositories, where organizations create and

maintain the resources and people can just look for the material they need. Other kind of solutions are the ones developed by some universities, based on datasets. They have implemented several applications to show useful information regarding a topic.

Moreover, regarding the cybersecurity educational area, not all trainings have the same pattern. Firstly, the traditional exercises based on software pilots are found, next to the virtual machine based ones. But, on the other hand, some new kind of training materials are emerging. They are based on virtual scenarios and on testing the participant skills. The target of this project is to obtain the metadata of the second mentioned, those which rely on virtual machines.

**SOLUTION ANALYSIS**

The system consists on the web scraping some information regarding the trainings allocated on the ENISA and SEEDLabs webs. Once the data is obtained, the access to it is provided by a browser interface and an API REST for developers.

The requirements are diverse but all of them look for the same goal, to favor the development of a strong, scalable, secure and useful system. The scope of this TFG is very large, so it is necessary to split it in several modules in order to get a better global approach.

The first module that has been studied is the data source. It has been done a deep analysis of the target webs and it has been decided to implement an own web scraping system in order to avoid the use of external software and to optimize the data extraction. Moreover, the extraction of the document terms has been done by some python libraries which allow the filter of words that don't contribute by themselves.

The second module is based on the way that the data is going to be stored. It has been decided to use a file system in which each folder will correspond to a training. Moreover, all the metadata extracted from the websites, will be written on a JSON file stored on the root of this filing system.

The last solution module is about giving the access to the data. After studying all the technologies capable of doing so, it has been chosen the one based on web application and the API REST for the developer users. To do so, the tool selected is Django, which allows an easy and fast development.

**DESIGN AND IMPLEMENTATION**

On this point, it will be described how the system has been designed and how it has been developed.

Starting by the file system, the web scraping, it has been mentioned before that each training has a corresponding folder on it. On this folder, despite it is not the goal, all the documents, like scripts or PDF's are going to be stored to analyze them later. Moreover, the file obtained by the training's documents analysis will be saved on its corresponding directory. The file system is generated by the script, which also extracts the data from the web. It first connects to the given url, and once it has obtained all the data, it stores it on a JSON file. Later, it creates the file system and saves it on the corresponding folder of the training's materials. The script has been coded on python3 using the "beautiful soup" next to the "requests" modules to ask for the metadata information and the "os" module to generate the file system.

Continuing by the terms analysis, another script is launched in order to analyze all the PDF's documents presented in each folder. So, the software walks through the directories' tree looking for these types of files and generating an output file, called "mainwordfile.txt", where all the relevant terms are included. Finally, this file is saved on its corresponding folder. The script has been also coded on python making use of the "nltk2", "pyPDF2" and "textract" modules to parse and analyze the documents' text.

Finally, the access module, which is divided on two pieces too, has been developed by Django, over python 3. Django provides a good functionality to do this split possible: the application concept. An application is a module of the web that can be exportable, so, it has been set 2 Django applications on the project.

The first one, related to the web interface, has a lot of views, where the server logic is coded. On this views, the basic and advanced search have been developed. This part, makes use of the default authentication mechanism that Django provides and extends it with the installation of the social_django module that allows to login with a google or GitHub account. The file "urls.py" specifies the url where a determined functionality can be found across the web.

The second application is related to the API REST access. On this part, a simple search is done. Including the corresponding code on the file "views.py", the API user will be able to make requests with an unique parameter related with the name of the metadata and the value that they want to find. The security on this process is given by the JWT

mechanism that requires to make a previous request with the credentials, in order to get a token to include on the following requests, in the Authorization header.

**DEPLOYMENT OF THE ENVIRONMENT.**

On this section, the requirements needed to deploy the environment are described. On the one hand, Python 3 is needed to be installed. This programming language, allows the use of virtual environments, which are very useful in order to install different module versions without taking care of the dependencies. On the other hand, despite it is not the goal of the TFG the fact of putting into production the server, some relevant aspects have been given to facilitate this task. For example, it has been mentioned that is preferable to use apache instead of NGINX as web server if the received traffic is not very high. Also, some tips about the domain name, the SSL certificate and regulatory affairs has been described.

**TESTS**

Some tests have been done in order to demonstrate the soundness and scalability of the system. These ones have targeted goals like the login, the search, the advanced search, the file system generation or the JSON file creation, and all of them have been positive.

**PROJECT MANAGEMENT**

The project management has been divided in three parts.

The first one, is regarding to the tool which has been used to manage the different versions that have been generated. GitHub is a software that allows to use an online repository to save the progress of the development and keep control of the changes that have been done.

The second one, is about the planning. The project began on January and will finish on July. On this period, activities based on learning new technologies, like Python or Django, tasks about coding the web scrapers, the web interface and the API REST have been made. Finally, the rest of the time are assigned especially to the elaboration of this document.

The third and the last one, the estimated budget. To do it, the personal and material costs have been calculated. Then, 25% of the benefits have been fixed. Finally, the IVA has been added, yielding a total project value of **12367.64 €.**

**REGULATORY FRAMEWORK**

On this section, all the regulatory affairs are treated. The new GDPR law is described and compared with the old LOPD. The European people have now the same rights and the data is more protected. Moreover, the copyright that affects the project is explained and concluded that the sources that have been used allow the distribution of their materials. Finally, the mechanism that some webpages implement against the web scrapers are mentioned. The file “robots.txt” describes if the server allows the use of this kind of software.

**CONCLUSIONS**

The last section tackles the conclusions obtained in the development of this project. Regarding the objective achievements, it can be concluded that all of them have been reached. About the personal conclusions, the feelings generated with the execution of this TFG have been positive since I have learnt many knowledges. Finally, some future lines have been added so that the scope of this project doesn’t die with the attainment of the fixed objectives.

# ANEXO II – ESQUEMA DEL SISTEMA DE FICHEROS

```
.
├── EnisaFiles
│   ├── Primer\ ejercicio
│   │   └── mainwordfile.txt
│   ├── Segundo\ ejercicio
│   │   ├── Presenting-_correlating_and_filtering_various_fe
│   │   └── mainwordfile.txt
│   └── Tercer\ ejercicio
│       ├── Mobile_Threats_Incident_Handling_Part_II-Student
│       └── mainwordfile.txt
├── SeedFiles
│   ├── Primer\ ejercicio
│   │   ├── Android_Rooting.pdf
│   │   └── mainwordfile.txt
│   ├── Segundo\ ejercicio
│   │   ├── Android_Repackaging.pdf
│   │   ├── MaliciousCode.smali
│   │   ├── MaliciousCode_Location.zip
│   │   ├── RepackagingLab.apk
│   │   └── mainwordfile.txt
│   └── Tercer\ ejercicio
│       ├── Buffer_Overflow.pdf
│       ├── call_shellcode.c
│       ├── exploit.c
│       ├── exploit.py
│       ├── mainwordfile.txt
│       └── stack.c
├── cysectrainingface
│   ├── APIfaceREST
│   │   ├── __init__.py
│   │   ├── __pycache__
│   │   │   ├── __init__.cpython-37.pyc
│   │   │   ├── admin.cpython-37.pyc
│   │   │   ├── models.cpython-37.pyc
│   │   │   ├── urls.cpython-37.pyc
│   │   │   └── views.cpython-37.pyc
│   │   ├── admin.py
│   │   ├── apps.py
│   │   ├── migrations
│   │   │   ├── __init__.py
│   │   │   └── __pycache__
│   │   │       └── __init__.cpython-37.pyc
│   │   ├── models.py
│   │   ├── tests.py
│   │   ├── urls.py
│   │   └── views.py
│   ├── cysectrainingface
│   │   ├── __init__.py
│   │   ├── __pycache__
│   │   │   ├── __init__.cpython-35.pyc
│   │   │   ├── __init__.cpython-37.pyc
│   │   │   ├── settings.cpython-35.pyc
│   │   │   ├── settings.cpython-37.pyc
│   │   │   ├── urls.cpython-35.pyc
│   │   │   ├── urls.cpython-37.pyc
│   │   │   ├── wsgi.cpython-35.pyc
│   │   │   └── wsgi.cpython-37.pyc
│   │   ├── settings.py
│   │   ├── urls.py
│   │   └── wsgi.py
│   ├── db.sqlite3
│   ├── manage.py
│   ├── socialauth
│   │   ├── __init__.py
│   │   ├── __pycache__
│   │   │   ├── __init__.cpython-35.pyc
│   │   │   ├── __init__.cpython-37.pyc
│   │   │   ├── urls.cpython-35.pyc
│   │   │   ├── urls.cpython-37.pyc
│   │   │   ├── views.cpython-35.pyc
│   │   │   └── views.cpython-37.pyc
│   │   ├── admin.py
│   │   ├── apps.py
│   │   ├── logs.txt
│   │   ├── migrations
│   │   │   └── __init__.py
│   │   ├── models.py
│   │   ├── tests.py
│   │   ├── urls.py
│   │   └── views.py
│   └── templates
│       ├── aboutus.html
│       ├── advancedsearch.html
│       ├── emptylist.html
│       ├── found.html
│       ├── foundlogic.html
│       ├── layout.html
│       ├── search.html
│       ├── showtraining.html
│       ├── socialauth
│       │   ├── login.html
│       │   └── logout.html
│       └── usefulinfo.html
├── data.json
├── enisascrapper.py
├── seedscrapper.py
└── textracting.py
```

# ANEXO III – REQUIREMENTS.TXT

```
argcomplete==1.8.2
beautifulsoup4==4.5.3
bootstrap4==0.1.0
bs4==0.0.1
certifi==2019.3.9
chardet==2.3.0
defusedxml==0.6.0
Django==2.1.7
django-sslserver==0.20
djangorestframework==3.9.2
djangorestframework-simplejwt==4.3.0
docx2txt==0.6
EbookLib==0.15
idna==2.8
lxml==4.3.3
nltk==3.4.3
oauthlib==3.0.1
Pillow==6.0.0
PyJWT==1.7.1
PyPDF2==1.26.0
python-pptx==0.6.5
python3-openid==3.1.0
pytz==2019.1
requests==2.22.0
requests-oauthlib==1.2.0
six==1.10.0
social-auth-app-django==3.1.0
social-auth-core==3.2.0
soupsieve==1.9.1
SpeechRecognition==3.6.3
textract==1.6.1
urllib3==1.25.3
wget==3.2
xlrd==1.0.0
XlsxWriter==1.1.8
```